NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

VLSI DESIGNS FOR PIPELINED FFT PROCESSORS

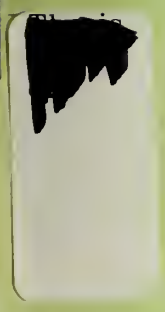
by

David Charles Stuart

June 1990

Thesis Advisor: Herschel H. Loomis, Jr.

Approved for public release; distribution is unlimited.



REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution/Availability of Report: Approved for public release; distribution is unlimited.	
2b Declassification/Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization: Naval Postgraduate School	6b Office Symbol (*if applicable*): EC	7a Name of Monitoring Organization: Naval Postgraduate School	
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000		7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	
8a Name of Funding/Sponsoring Organization	8b Office Symbol (*if applicable*)	9 Procurement Instrument Identification Number	
8c Address (*city, state, and ZIP code*)		10 Source of Funding Numbers	
		Program Element No	Project No / Task No / Work Unit Accession No

11 Title (*include security classification*): VLSI DESIGNS FOR PIPELINED FFT PROCESSORS (Unclassified)

12 Personal Author(s): David Charles Stuart

13a Type of Report	13b Time Covered	14 Date of Report (*year, month, day*)	15 Page Count
Master's Thesis	From To	June 1990	201

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	VLSI multipliers, VLSI adders, pipelined arrays, floating-point multipliers, floating-point adders, FFT processors

19 Abstract (*continue on reverse if necessary and identify by block number*)

A system of custom cell building blocks utilizing scaleable CMOS technology is described. The cells are designed to support the high speed, pipelined addition, subtraction, and multiplication operations necessary in a cyclic spectral analyzer or other applications involving the FFT. The cells are structured in such a manner as to permit a designer to tailor the bit-length of the operations and the number of pipeline stages used. Both fixed and floating operations are supported by the system. The size and performance characteristics of devices produced using the cells are compared with previously produced Genesil Silicon Compiler pipelined designs. The appendix contains designs of a 16-bit mantissa, 12-bit exponent floating point multiplier and adder produced from the standard cells. If fabricated in 1.2-μ feature size technology, the theoretical maximum clock speed and throughput rate is 102 MHz with an asymmetric clock and 61 MHz using a symmetric clock waveform. Devices with clock speeds up to 178 MHz are possible if the number of logic cells between a pipeline stage is reduced to one.

20 Distribution/Availability of Abstract	21 Abstract Security Classification	
☒ unclassified unlimited ☐ same as report ☐ DTIC users	Unclassified	
22a Name of Responsible Individual: Herschel H. Loomis, Jr.	22b Telephone (*include Area code*): (408) 646-3124	22c Office Symbol: ECLm

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete

Approved for public release; distribution is unlimited.

VLSI Designs for Pipelined FFT Processors

by

David Charles Stuart
Lieutenant, United States Navy
B.S., Southern Illinois University at Carbondale, 1979

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
ELECTRICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

ABSTRACT

A system of custom cell building blocks utilizing scaleable CMOS technology is described. The cells are designed to support the high speed, pipelined addition, subtraction, and multiplication operations necessary in a cyclic spectral analyzer or other applications involving the FFT. The cells are structured in such a manner as to permit a designer to tailor the bit-length of the operations and the number of pipeline stages used. Both fixed and floating operations are supported by the system. The size and performance characteristics of devices produced using the cells are compared with previously produced Genesil Silicon Compiler pipelined designs. The appendix contains designs of a 16-bit mantissa, 12-bit exponent floating point multiplier and adder produced from the standard cells. If fabricated in 1.2-μ feature size technology, the theoretical maximum clock speed and throughput rate is 102 MHz with an asymmetric clock and 61 MHz using a symmetric clock waveform. Devices with clock speeds up to 178 MHz are possible if the number of logic cells between a pipeline stage is reduced to one.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. CYCLIC SPECTRAL ANALYSIS 1
B. PIPELINING CIRCUITS FOR HIGH PERFORMANCE 4
C. APPLICATION SPECIFIC INTEGRATED CIRCUIT DESIGN .. 7
D. THESIS GOALS AND ORGANIZATION 10

II. FULL-CUSTOM SCALEABLE CMOS 12
A. CMOS VLSI DESIGN PRINCIPLES 12
B. SCALING CMOS DESIGNS 12
C. FULL-CUSTOM DESIGN TOOLS 13

III. NUMBER SYSTEMS AND ALGORITHMS 17
A. NUMBER SYSTEMS AND ARITHMETIC OPERATIONS 17
B. FLOATING POINT NUMBER SYSTEMS 21
C. FLOATING POINT MULTIPLICATION 26
D. FLOATING POINT ADDITION 28
E. OVERFLOW, UNDERFLOW, AND ROUNDING 30

IV. HARDWARE 33
A. ADDER DESIGNS 33
B. LATCH LAYOUTS 38
C. MULTIPLIER DESIGNS 42
D. OTHER CELL FEATURES. 47

V. DESIGN COMPARISONS 61

VI. CONCLUSIONS AND RECOMMENDATIONS 68
A. SUMMARY AND CONCLUSIONS 68

B. RECOMMENDATIONS 68

APPENDIX A. SPICE SIMULATION EXAMPLE 70

APPENDIX B. STANDARD CELL DESCRIPTIONS AND LAYOUTS 77
A. EXPONENT ADDITION FUNCTION CELLS 77
B. MULTIPLICATION FUNCTION CELLS. 104
C. EXPONENT SUBTRACTION FUNCTION CELLS. 151
D. MANTISSA ALIGNMENT AND SELECTION 188

LIST OF REFERENCES 192

INITIAL DISTRIBUTION LIST 194

I. INTRODUCTION

A. CYCLIC SPECTRAL ANALYSIS

Cyclic spectral analysis is a fundamental tool for the study of periodicity in signals and systems [Ref. 1: p. ii]. Signal detection, modulation recognition, and signal parameter estimation are only a few of the potential uses of this relatively new analysis technique. A particularly important application of cyclic spectral analysis is the study of modulated signals. As an example, consider results derived in Ref. 2. Figure 1 on page 2 contains the plots of four different phase-shift keyed (PSK) signals. The magnitude of the cyclic spectrum is plotted as the height above a bi-frequency plane defined by f and α where f is the spectral frequency and α is called the cyclic frequency. Notice that the four signals have identical power spectral density functions ($\alpha = 0$), but have highly distinct spectral correlation functions.

Several methods and algorithms exist for computing the cyclic spectrum of signals [Ref. 3 : pp. 1-17]. Figure 2 on page 3 (from Ref. 4) shows a block diagram of one such algorithm using a time-averaging method. The computation requires:

(a) prefiltering using overlapping window Fast Fourier Transforms (FFT's);

(b) multiplication by complex phase terms; and

(c) performing another, longer FFT.

These results cover only a small portion of the bi-frequency plane. Many such overlapping computations must be performed to cover the entire plane. This computational complexity, which far exceeds that of conventional spectral analysis, presently limits the use of the cyclic spectrum as a signal and systems analysis tool.

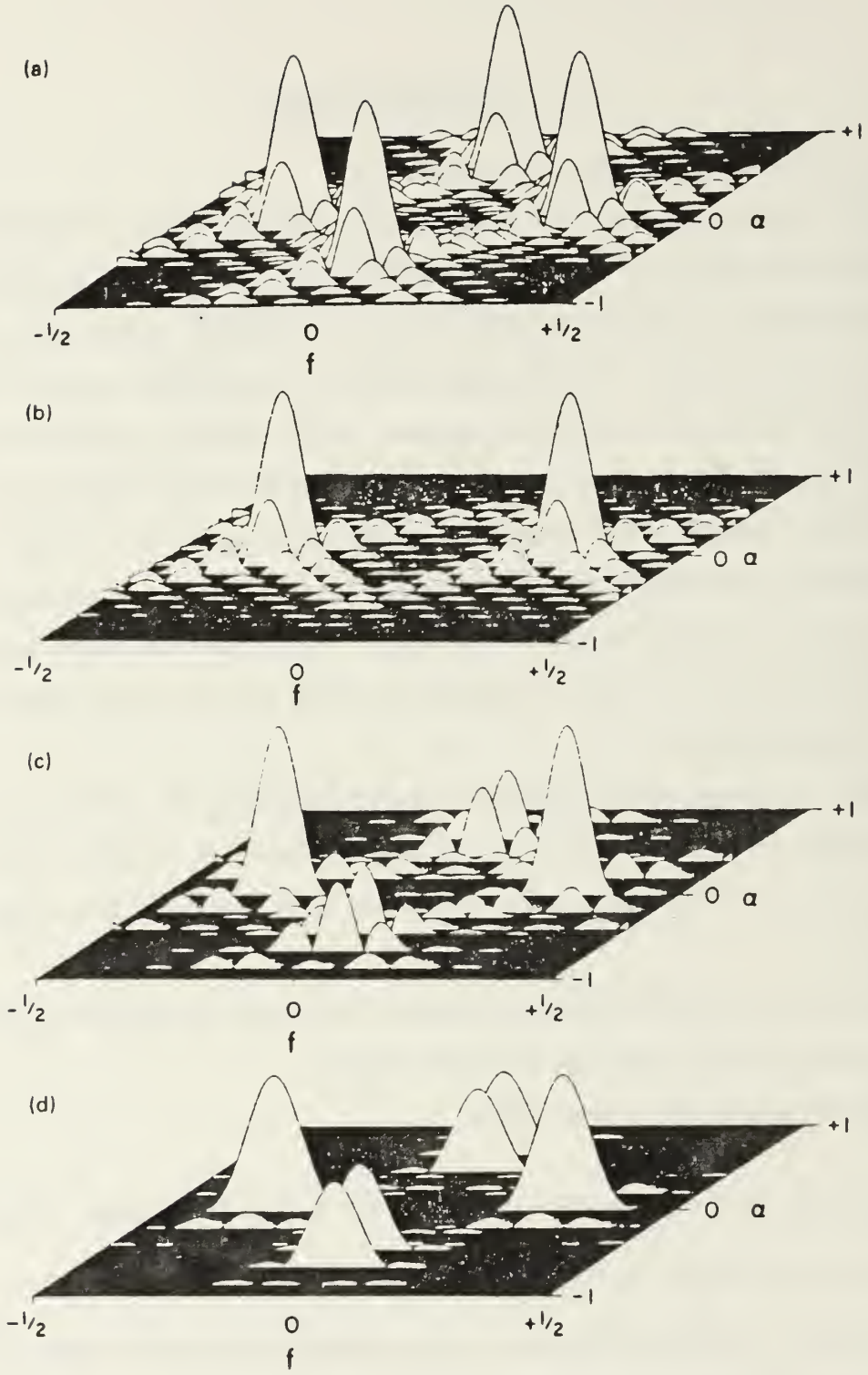


Figure 1. Cyclic spectrum magnitudes of four PSK signals.

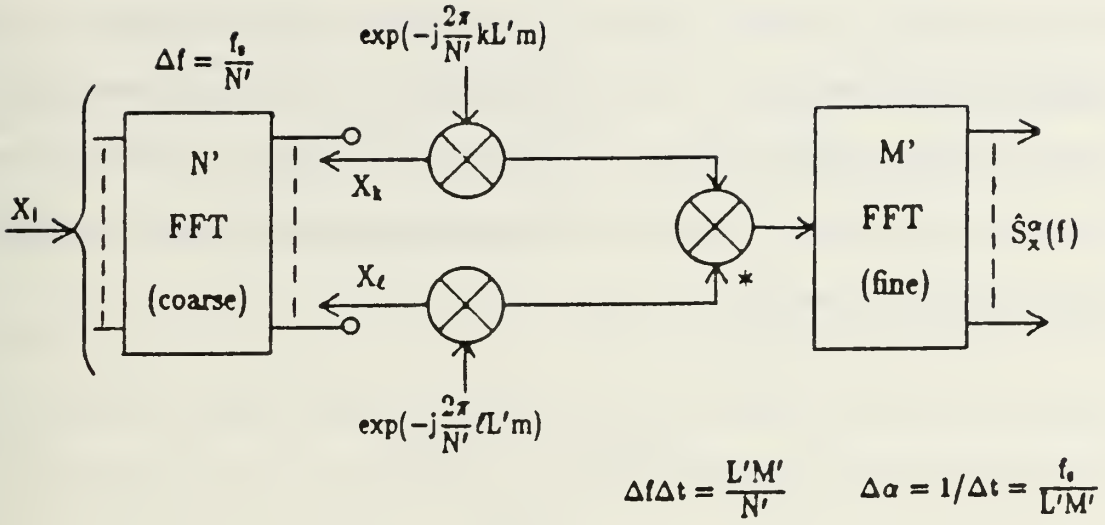


Figure 2. Time-averaging method

The basic operations required to compute the cyclic spectrum such as Fourier transformation and product modulation are common to most signal processing algorithms. However, the extreme number of these operations required to yield meaningful results prove taxing to general-purpose computers. A wide range of applications, including applied research in the field of cyclic spectral analysis would open up if cyclic spectral analysis algorithms could be computed more rapidly. One way of computing these algorithms more rapidly is to use computational systems that are specifically designed for digital cyclic spectral analysis. [Ref. 3: p. 2]

The mathematical operations required to compute the cyclic spectrum are multiplication, addition and subtraction. The cyclic spectral algorithm's use of structured computations such as the FFT may make implementation in very-large-scale integrated (VLSI) circuit devices practical if many mathematical

operations can be performed on a single chip. As an example, a radix-2 FFT decimation in time "butterfly" (Figure 3) requires one complex multiply and two complex addition operations. This is equivalent to four multiplications and six additions of real numbers [Ref. 5]. If such a set of computations could be performed on a single chip at sufficiently high rates, a near-real-time cyclic spectral analyzer might be practical.

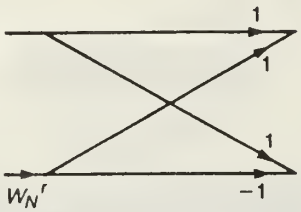


Figure 3. FFT "butterfly"

B. PIPELINING CIRCUITS FOR HIGH PERFORMANCE

The basic idea behind a pipelined design is quite natural. The name *pipeline* stems from the analogy with petroleum pipelines in which a sequence of products is pumped through a pipeline. Figure 4 on page 5 shows a sequential process made up of "N" discrete cascaded sub-processes. We can apply this analogy to an electronic circuit made up of cascaded logic stages. The maximum delay through the circuit is the sum of the maximum delays through each logic stage and can be expressed as:

$$T_{D_{\max}} = t_{d_1} + t_{d_2} + \ldots + t_{d_N}, \tag{1}$$

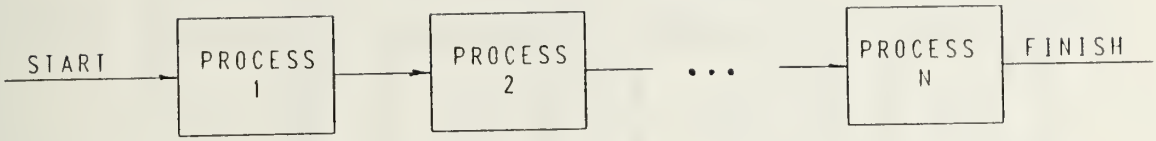


Figure 4. An N-step sequential process.

where $T_{D_{max}}$ is the maximum total circuit delay and $t_{d_1}, t_{d_2}, \ldots, t_{d_N}$ are the delays of the individual stages of logic. In using a circuit of this type, inputs usually must be held constant until the output is stable. The maximum rate at which this type of circuit can perform operations is, therefore,

$$R_{in_{max}} = 1/T_{D_{max}}. \tag{2}$$

Figure 5 on page 6 shows the circuit with clocked registers added between the logic blocks. These registers are used to save the state of the preceding logic block at periodic intervals, and then to input that state into the next set of logic. The minimum clock period that could be used in this circuit is

$$T_{C_{min}} = t_{d_{max}} + t_{d_{register}}, \tag{3}$$

where $t_{d_{max}}$ is the maximum delay of the slowest logic block and $t_{d_{register}}$ is the total delay associated with a register. If a design breaks up the operation into logic blocks with similar delay times, and the delay of a register is small compared with $T_{d_{max}}$, the input rate of the circuit can be significantly raised over that of an

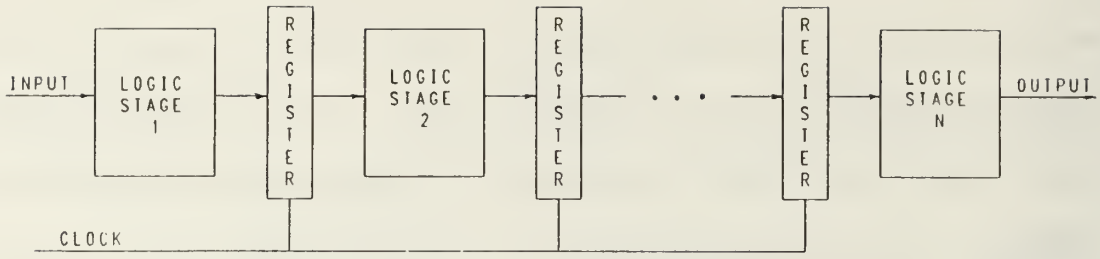


Figure 5. Pipelined process

unpipelined circuit. Notice that the total delay (sometimes referred to as *latency*) through the circuit is now

$$T_D = N \bullet T_C, \tag{4}$$

where N is the number of stages and T_C is the clock period.

This pipelining principle is easily applied to mathematical operations. For example, consider Figure 6 on page 7, a four-bit ripple-carry adder made up of four one-bit full adders. All the inputs (A's and B's) must be held at a fixed logic level as the carry "ripples" through the adders. This adder can be pipelined by adding registers as in Figure 7 on page 8. This adder could operate with higher data input rates, but the design has several potential disadvantages.

First, the answer from one addition operation is not available until several clock periods after the next addition can be started. If the next addition to be performed requires a previous output, it cannot be started until that previous output is available. This could cause the pipeline to "empty" and defeats the purpose of pipelining. It may even result in slower overall operations, since the

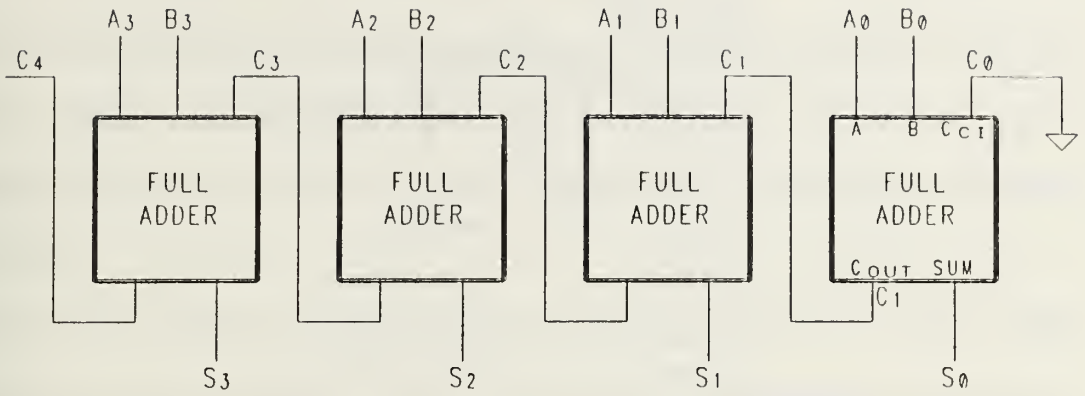


Figure 6. Four-bit ripple-carry adder

overall delay of the circuit is increased by pipelining. Pipelining will only increase the throughput of a system if the computations can be structured in a way to keep the pipeline "full". Fortunately, an FFT-type algorithm can be structured in just such a way [Ref. 6].

Secondly, the pipelined system is physically larger and more complex. Not only does it have all the logic required of the unpipelined circuit, it also requires registers and at least one clock. This extra space requirement can be very significant and could limit the use of pipelining in some applications.

C. APPLICATION SPECIFIC INTEGRATED CIRCUIT DESIGN

Application specific integrated circuit (ASIC) designs have become popular in the military. These circuits, custom designed for a particular application are especially useful in specialized, high performance systems. Traditional methods of ASIC design include full-custom design, gate-array circuit design, and standard cell circuit design [Ref. 7: p.38]. Silicon compilation is the newest method of ASIC design and allows the designer a higher degree of flexibility and feedback

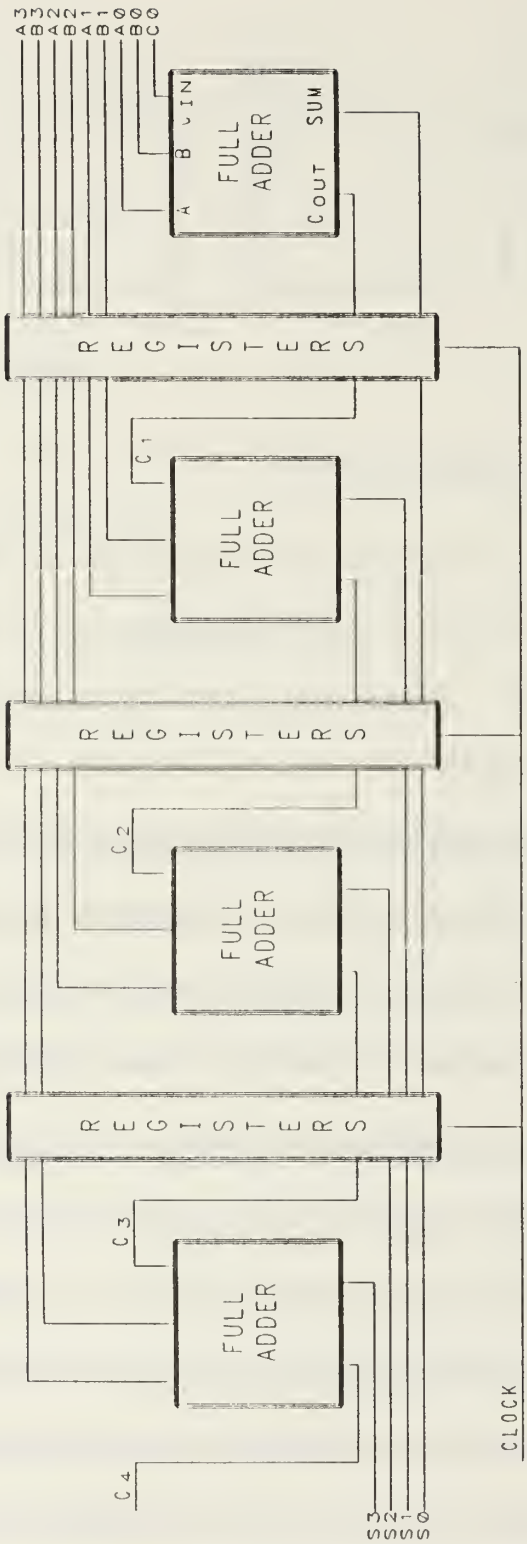


Figure 7. Pipelined four-bit adder

than previously available. The silicon compiler works from a high-level description of the circuit that allows the designer to perform successive design iterations quickly and efficiently, providing the designer rapid access to key parameters such as chip size, power consumption, and timing constraints [Ref. 8].

The Genesil Silicon Compiler used at the Naval Postgraduate School (NPS) in Monterey allows a designer to easily create complex circuits using devices from the compiler library. This design procedure requires little knowledge of VLSI circuit design on the transistor level. Two theses done at NPS, References 9 and 10, provide excellent background on the system and its capabilities.

Several pipelined adders and multipliers designs have been produced at NPS using the Genesil Silicon Compiler. Reference 9: pp. 46-61 documents a custom 16-bit integer adder that reduced the maximum propagation delay by slightly less than a factor of three. This increase in performance was achieved with a significant sacrifice in circuit area; the pipelined version took 23 times the chip surface area of the unpipelined adder. Pipelined multiplier designs also produced large increases in chip surface area requirements [Ref. 9: pp. 62-81].

Since the physical size of IC layouts limit the number of mathematical operations that can be performed on a single chip, the large increase in chip area necessary to pipeline designs using the Genesil Silicon Compiler could limit its usefulness in designing chips for a cyclic spectral analyzer. Other design methods, such as full-custom, might be more appropriate.

Full-custom design of integrated circuits is a time-consuming method that requires extensive programming for simulation and timing analysis of the VLSI

design prior to fabrication. Full-custom design is normally used by IC manufacturers producing large quantities of standard off-the-shelf type chips such as microprocessors. With a full-custom chip, the designer has the capability to control the physical arrangement at the lowest level. The designer using Genesil can only rearrange the compiler library devices. A full-custom design should be able to produce chips which use less area than chips developed by Genesil, but the only way to determine the layout size is to first produce a full-custom design.

D. THESIS GOALS AND ORGANIZATION

In order to make appropriate design choices to produce pipelined VLSI components for a cyclic spectral analyzer certain information is needed such as:

- Is it feasible to use full-custom design to produce pipelined adders, multipliers and subtracters?
- If feasible, do the full-custom devices offer significant advantages over Genesil-produced designs?
- Can enough mathematical operations be put on a single IC chip to perform an FFT "butterfly" operation?
- If so, what feature size will be necessary?

This thesis answers these questions by developing a full-custom VLSI cell *system* that can be used to produce pipelined adders, subtracters, and multipliers appropriate for a cyclic spectral analyzer. This system will be capable of producing both fixed and floating point mathematical operations to enable a designer to more appropriately evaluate the size and accuracy tradeoffs inherent in each type format.

Chapter II describes the CAD design tools and programs available at NPS for full-custom design. The principles of scaleable CMOS devices that make them appropriate choices for cyclic spectral analyzer components are also

discussed. Chapter III describes the features of the number systems used, including floating point operations. Chapter IV describes the basic adder and multiplier cell components. Other specialized devices are also documented. Chapter V documents the size and performance specifications for fixed and floating point operations produced from the cell designs. Appendix A contains sample Spice inputs and results. Appendix B contains standard cell descriptions and layouts.

II. FULL-CUSTOM SCALEABLE CMOS

A. CMOS VLSI DESIGN PRINCIPLES

The past few years have seen a rapid shift in the technology of choice for high-complexity digital microelectronics from nMOS to Complementary Metal Oxide Silicon (CMOS) [Ref. 11: p. ix]. This shift has occurred because CMOS offers high performance with low power consumption; CMOS sinks current from the power line only when logic transitions occur [Ref. 12: p.51]. CMOS designs also scale extremely well to small feature sizes, enhancing their speed and decreasing the chip area used [Ref. 11 : p. 150].

A VLSI chip with hundreds of thousands of transistors can be an extremely complex device. The role of design aids and strategies is to reduce this complexity and assure the designer a working product [Ref. 11 : p. 238]. Choosing appropriate architectures is a necessary step in the simplification process. If a structure can truly be decomposed into a few types of simple substructures or building blocks, which are used repetitively with simple interfaces, the principles of modularity and hierarchy can be applied [Ref. 13].

B. SCALING CMOS DESIGNS

As CMOS processes are improved and device dimensions are reduced, the performance of a CMOS design changes. Many times, prototype devices or subsystems are designed, fabricated, and tested using relatively inexpensive "large" feature sizes [Ref. 12: p. 53]. Later, these prototype designs may be incorporated into a production device which is fabricated with smaller dimensions. The

effects that the reduced dimensions have on the electrical behavior of the device are important considerations in VLSI design and the final choice of feature size.

First-order MOS scaling theory is based on a "constant field" model formulated by R. H. Dennard [Ref. 11: p. 150]. A device is "scaled" by applying a dimensionless factor α to:

- all dimensions, including those vertical to the surface,
- device voltages, and
- the concentration densities.

The resultant effects of this first-order scaling process are illustrated in Table 1 on page 14.

C. FULL-CUSTOM DESIGN TOOLS

Computer-Aided Design (CAD) tools and programs are used by a designer to simplify the design process and make large, complex designs feasible. Performance predictions, design validation, and checking are also enhanced (or made possible) by the use of these design tools.

The VLSI design facilities available at NPS include a custom VLSI design tools "package" of programs put together by the CS Division of the Department of EECS, University of California, Berkeley [Ref. 14: p. 1]. The package consists of about twenty programs for designing and analyzing VlSI circuits. Reference 14 contains descriptions of these programs, and tutorials on the graphical layout editor. Familiarity with this design package is necessary for anyone wishing to use cells developed in this thesis. As an overview, the following are some of the design tools applicable:

Crystal A timing analyzer that assists the designer in identifying performance problems in a design.

Table 1. **FIRST-ORDER SCALING EFFECTS ON MOS DEVICES** (from Ref. 11: p. 152)

PARAMETERS:	SCALING FACTOR:
Length; L	$1/\alpha$
Width; W	$1/\alpha$
Gate oxide thickness; t_{ox}	$1/\alpha$
Junction depth; X_j	$1/\alpha$
Substrate doping; $N_a(\vee d)$	α
Supply voltage; V_{DD}	$1/\alpha$
Electric field across gate oxide; E	1
Depletion layer thickness; d	$1/\alpha$
Parasitic capacitance; WL/t_{ox}	$1/\alpha$
Gate delay; (Vc/I)	$1/\alpha$
DC power dissipation; P_s	$1/\alpha^2$
Dynamic power dissipation; P_d	$1/\alpha^2$
Power-speed product	$1/\alpha^3$
Gate area	$1/\alpha^2$
Power density; (VI/A)	1
Current density; (I/A)	α
Transconductance; g_m	1

Esim An event driven logic-level simulator developed at MIT and distributed with their permission.

Ext2Sim Part of the Magic suite of programs. Used for converting the output of Magic's hierarchical extractor into a form usable by other tools such as Esim, Crystal, and Spice.

Magic The graphical layout editor.

The graphical layout editor, Magic, is the heart of the CAD design package. CMOS designs made using Magic's SCMOS technology are flavor-less and

scaleable; they may be fabricated in either N-well or P-well technology in a variety of feature sizes. The *lambda* units used in Magic are dimensionless; MOSIS currently supports fabrication at the 1.5 microns/lambda, 1.0 microns/lambda, and 0.7 microns/lambda scale factors [Ref. 14: p. 285]. These scale factors correspond to 3.0, 2.0, and 1.2-micron feature sizes respectively. Other scale factors are expected to become available in the future.

Using Magic, a designer "draws" rectangles of polysilicon, diffusion, metal, and various contacts to generate VLSI layouts. Magic supports hierarchical design methods; layouts that are collections of cells and sub-cells are easily generated. Magic also automatically checks designs to verify that design rules have been adhered to. Rule violations are displayed on the screen in the vicinity of the error [Ref. 14: p. 189]. Figure 8 shows an example layout annotated with corresponding layer labels.

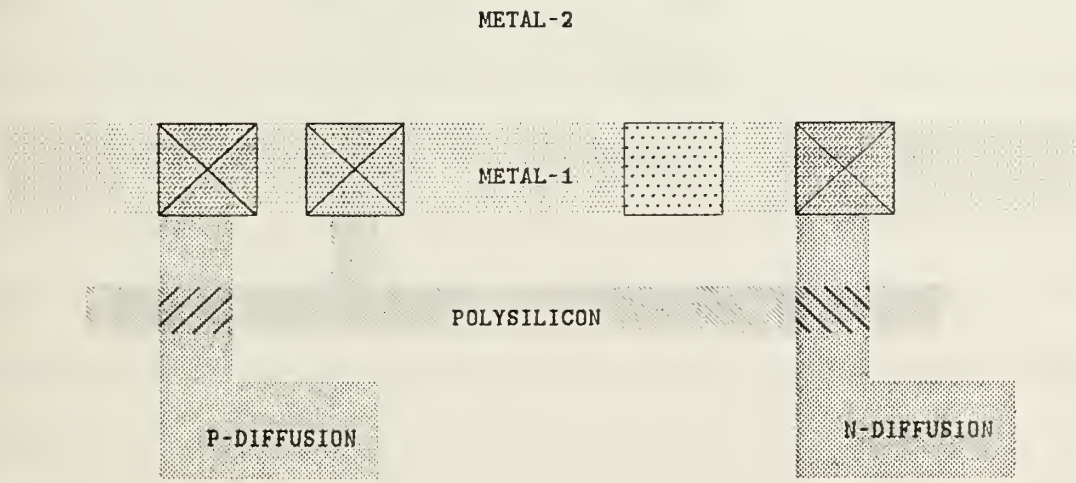


Figure 8. Legend for layout layers and contacts.

Spice, a general-purpose circuit simulation program for non-linear analysis, was also used for verification and optimization of low-level circuit designs produced in this thesis. Spice has built-in models for semiconductor devices; a user need only specify the pertinent model parameters [Ref. 15]. Spice performs DC and transient analysis at various temperatures selected by the designer. Spice utilizes matrix equations relating voltages, currents, and resistances. This type of simulation is characterized by high accuracy but long simulation times. Simulation time is typically proportional to n^2 where n is the number of nonlinear devices in the circuit. It is unrealistic to use this type of program for the verification of large VLSI chips. For large numbers of transistors switch-level-type simulators are generally used to verify logic and wiring paths. [Ref. 11: p. 255]

III. NUMBER SYSTEMS AND ALGORITHMS

A. NUMBER SYSTEMS AND ARITHMETIC OPERATIONS

Three number systems were considered for use in arithmetic operations:

- sign-magnitude,
- ones' complement, and
- two's complement.

The addition process, when using a ones' complement number system, requires adding the carry-out of the most significant bit back into the carry-in of the least significant bit. Letting this carry "ripple through" an adder the second time is costly in terms of overall delay. It is also undesirable in pipelined adders from a layout size consideration; performing the addition process twice requires more hardware in a pipelined system. For these reasons, use of the ones' complement number system was rejected.

The sign-magnitude system uses the most significant bit to represent the sign of the number; all other bits represent the magnitude. Customarily, a *1* in the sign bit position represents a negative number and a *0* represents a positive number. That convention is used in the system described in this thesis. The magnitude of an N-bit binary number represented using a sign-magnitude system is given by

$$V_{magnitude_{fixed-point}} = \sum_{i=0}^{N-2} b_i \cdot 2^{i-p}, \tag{5}$$

where b_i represents the *ith* bit ($i = 0, 1,...,N-1$), and p identifies the location of the radix point (specifically, the number of bits to the right of the radix point). The sign-magnitude system appears well-suited for use in the multiplication process. The magnitude of the product depends only on the magnitude of the multiplier and the magnitude of the multiplicand; the sign of the product depends only on the sign of the multiplier and multiplicand.

The two's complement number system represents both positive and negative numbers. To negate a value, the value is subtracted from 2^N [Ref. 16: p. 33]. The value of a particular representation is given by:

$$V_{twos-complement_{fixed-point}} = -b_{N-1} \cdot 2^{N-p-1} + \sum_{i=0}^{N-2} b_i \cdot 2^{i-p}, \qquad (6)$$

where N, p, and i are defined as described in the sign-magnitude system. The two's complement has the advantage of representing one more value than an equally sized sign-magnitude system. The value zero only has one representation in two's complement, vice the two representations in a sign-magnitude system. Also, subtraction can be accomplished by adding the two's complement of a number. This provides advantages in a system performing addition and subtraction of numbers which can be either positive or negative. To perform addition, simply add the numbers regardless of whether either or both are positive or negative. The answer will be correct (assuming no overflow, discussed later).

The addition of two numbers, A and B, in a sign-magnitude system is much more complex; several cases must be considered as listed below with the sign bit explicitly shown inside the parenthesis:

1. (+A) + (+B),
2. (-A) + (+B),
3. (+A) + (-B), and
4. (-A) + (-B).

The subtraction process has similar variations. Cases 1 and 4 can be accomplished by adding the magnitudes and setting the sign bit positive if A and B are positive and negative if A and B are both negative. Cases 2 and 3 are more complex. Either case could be accomplished by subtracting B from A *and* subtracting A from B, then choosing the "correct" magnitude and sign based on the relative magnitude and sign of A and B. This is a complex set of procedures that could require performing each addition (or subtraction) three "ways" simultaneously, and then choosing the correct answer. This procedure would use significantly more chip area than a two's complement number system.

Subtraction in a two's complement number system can be accomplished by taking the two's complement of the subtrahend and adding it to the minuend. The two's complement of a number can be obtained by inverting all bits and adding a 1 to the least significant bit [Ref. 16: p. 35]. Figure 9 on page 20 shows a possible gate-level logic design which will accomplish this complement and increment operation. Notice that the conversion process will "ripple" from right to left. Also note that the least significant bit of the operand is available immediately. It is not necessary to perform this conversion process *prior to* a

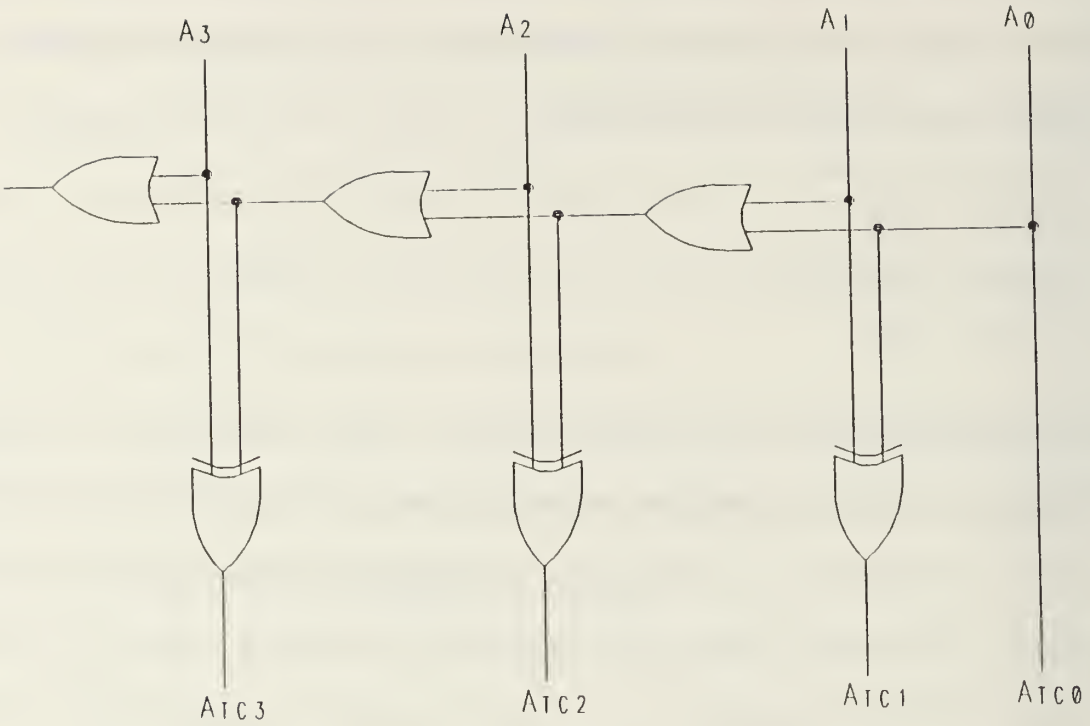


Figure 9. Two's complement converter

subtraction; simply feed the complement (invert each bit individually) of the subtrahend into an adder and assert the carry-in of the adder. This is simple and fast; in CMOS, inverters are small, rapidly operating devices. Due to the overall simplicity, size constraints, and the number of additions and subtractions in an FFT algorithm, the two's complement system was selected for use in the addition and subtraction cell designs produces in this thesis.

The choice of a number system for the multipliers in the FFT butterfly still leaves two options:

1. Use a sign magnitude system, converting from two's complement prior to the multiply and then converting back to two's complement after the multiply; or,

2. Use an algorithm that correctly computes the product of two two's complement numbers.

Reference 17 documents a two's complement parallel array multiplication algorithm. Other algorithms, such as Booth's [Ref. 16: p. 90], are also available. All these algorithms require complicated interconnection patterns which increase the complexity of a multiplier and require more space on the chip [Ref. 18: p. 35]. Keeping in mind the design goal of producing a structure which uses simple repetitive substructures, the sign-magnitude system was chosen for the multiplier design.

B. FLOATING POINT NUMBER SYSTEMS

Fixed point number systems are frequently used in digital signal processing applications. If values are scaled as they enter, a system can be designed to ensure that intermediate values are sufficiently represented by the number of bits in the system [Ref. 16: p. 37]. In a radix-2 FFT butterfly with both a real and an imaginary part in each input, the maximum any output can increase in a single stage is

$$1 + \sin(45°) + \cos(45°) = 2.414213562. \tag{7}$$

Many FFT implementation schemes take this maximum growth into account and compensate by scaling each stage to prevent "overflow" [Ref. 19: pp. 75-76]. By implementing such scaling practices and carefully examining accuracy requirements, a system designer may preclude the necessity of the wider range of data representation available in a floating point number system. This thesis produces designs in both fixed and floating point formats to allow a designer flexibility in

choosing which type of number system to use based on accuracy requirements, size of the VLSI implementations, and number of stages of delay in the pipeline.

To specify a floating point number, seven different pieces of information are required:

- base of the system; r_b,
- sign of the mantissa; S_m,
- magnitude of the mantissa; M_m,
- base of the mantissa; usually also r_b,
- sign of the exponent; S_e,
- magnitude of the exponent; M_e, and
- base of the exponent; r_e.

The value of the number represented, V_{fpn}, is

$$V_{fpn} = (-1)^{S_m} \bullet M_m \bullet (r_b)^{V_e}, \tag{8}$$

where V_e, the value of the exponent, is determined from S_e, r_e, and M_e [Ref. 16: pp. 42-45]. While the representation shown above implies use of a sign-magnitude system for expressing the values of the mantissa and exponent, this is not necessary. Other types of number systems are used quite frequently.

The excess number system is one such number system. An excess code is produced by purposely adding an "excess" to the value to be represented. The resultant bit pattern is then stored or used as required. One of the most prevalent uses of excess codes is to store exponents in floating point number systems [Ref. 16: p. 38]. If S represents the excess code value that will be stored or used, V the true value of the number, and E the excess, then the relationship can be expressed as

$$S = V + E. \tag{9}$$

Table 2 on page 24 (from Ref. 16: p. 55) identifies a number of machines and the floating point system they use. Note that most systems use excess coded exponents which will represent the smallest number (most negative value of exponent) with an excess coded exponent equal to .000...0. In all cases where excess coded exponents are used, the coded exponent will never be negative. This allows various machines to use a special code for an "exact" zero. This practice also makes some floating point operations less complex at the expense of other operations.

In addition to the previously discussed characteristics, most floating point number systems use a process called normalization which makes an assumption as to the location of an "implied" radix point. This eliminates the need to code and store radix point location. The floating point number system used in this thesis assumes the most significant bit of the mantissa (excluding the sign bit) is the first bit to the right of the radix point. This first bit is usually never allowed to be a *0* (exceptions discussed later). This means that the only legal values of the magnitude of the mantissa is a fraction between 1/2 (assuming $r_b = 2$) and almost *1*. For example, consider a sign-magnitude system which uses a 4-bit mantissa. Showing the sign bit explicitly, the largest legal mantissa would be

$$01111 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = \frac{15}{16}. \tag{10}$$

The smallest positive mantissa is

Table 2. FLOATING POINT INFORMATION SYSTEMS

System	Word Size (# Bits)	Exponent r_b	Exponent # Bits	Exponent Code	Mantissa # Bits	Mantissa Repre.	Mantissa Code
Burroughs B6700/7700	48	8	7	SM	39	Int	SM
CDC 7600	60	2	11	Ex 1024	48	Int	1's C
DEC — single	32	2	8	Ex 128	24	Fra	SM
DEC — double	64	2	8	Ex 128	56	Fra	SM
Honeywell 8200	48	2 or 10	7	Ex 64	40 (base 2) 20 (base 10)	Fra Fra	 SM
IBM — single	32	16	7	Ex 64	24	Fra	SM
IBM — double	64	16	7	Ex 64	56	Fra	SM
IEEE — single	32	2	8	Ex 127	24	Fra	SM
IEEE — double	64	2	11	Ex 1023	53	Fra	SM
Cray	64	2	15	Ex 16384	48	Fra	SM

Int = Integer representation

SM = Sign/magnitude

Fra = Fractional

1's C = One's complement

Ex = Excess code

$$01000 = \frac{1}{2} \tag{11}$$

The largest (most negative) mantissa is

$$11111 = (-1) \cdot (\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}) = -\frac{15}{16}. \tag{12}$$

This system of normalization will also work with a two's complement number system with some modifications. Consider the following examples with an implied radix point to the right of a "sign" bit:

$$01111 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = \frac{15}{16}, \tag{13}$$

$$01000 = \frac{1}{2}, \tag{14}$$

$$11000 = -1 + \frac{1}{2} = -\frac{1}{2}, \tag{15}$$

$$11111 = -1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = -\frac{1}{16}. \tag{16}$$

Notice that the convention of requiring the first bit to the right of the radix point to be a 1 has resulted in a shift in the range of magnitude of the value represented. If the convention is modified to require a negative two's complement number to have a zero as the first bit to the right of the radix point, the correspondence between positive and negative numbers is more appropriate. Consider the following two's complement examples:

$$10001 = -1 + \frac{1}{16} = -\frac{15}{16}, \tag{17}$$

$$10111 = -1 + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = -\frac{9}{16}, \tag{18}$$

$$10000 = -1. \tag{19}$$

This thesis uses a sign-magnitude system for multiplication and a two's complement system for addition. Such a mixing of systems must allow for conversion from one to another. The above examples show two specific instances where a legal representation in one system has no corresponding legal representation in the other system. To compensate for this, these two cases must be sensed, specifically:

1. In conversion from two's complement to sign-magnitude, if a 1000...0 is sensed, the two's complement number is "sign-extended" and shifted one bit to the right before conversion and the exponent value is increased by 1.
2. In conversion from sign-magnitude to two's complement, if a 11000...0 is sensed, the number is shifted one bit to the left and the exponent is decreased by 1.

The floating point number system developed in this thesis uses a base of 2 with the exponent using an integer two's complement representation. In multiplication, a sign-magnitude number system representation is used in the mantissa; for addition and subtraction, a two's complement number system is used in the mantissa. Conversion between the two systems is performed as required.

C. FLOATING POINT MULTIPLICATION

Floating point multiplication of two numbers, using a sign-magnitude system, can be stated as

$$
\begin{aligned}
A &= B \times C \\
&= ((-1)^{S_{m_b}} \bullet M_{m_b} \bullet r_b^{V_{e_b}})((-1)^{S_{m_c}} \bullet M_{m_c} \bullet r_b^{V_{e_c}}) \\
&= (-1)^{(S_{m_b} + S_{m_c})} \bullet (M_{m_b} \bullet M_{m_c}) \bullet r_b^{(V_{e_b} + V_{e_c})}.
\end{aligned} \tag{20}
$$

Thus, the product mantissa is the product of the multiplier and multiplicand mantissas; the product exponent is the sum of the exponents. The sign bit can be easily obtained in VLSI circuitry; it is simply the "exclusive-or" of the sign bits. Note that the above three operations can be performed simultaneously; they do not depend on each other.

In any floating point operation using normalized operands the normalization of the result must be checked and adjusted to meet system requirements. This "post normalization" is relatively simple following a multiplication. Consider the product of the largest legal mantissas:

$$
\begin{array}{r}
0.1111 \\
\times\ 0.1111 \\
\hline
0.1110
\end{array} \tag{21}
$$

In this case, the mantissa is properly aligned; no post normalization is required. With the other extreme, the product of the smallest legal mantissas:

$$
\begin{array}{r}
0.1000 \\
\times\ 0.1000 \\
\hline
0.0100
\end{array} \tag{22}
$$

The mantissa must be shifted one bit to the left. This normalization requires subtracting a 1 from the exponent to complete the operation.

D. FLOATING POINT ADDITION

Compared to a multiplication, floating point addition is a much more complex operation. Figure 10 on page 29 [Ref. 16: p. 111] shows a block diagram for the addition process. First, the exponents are compared. The largest exponent is saved, and the difference in the exponents is used to shift the mantissa of the number with the smallest exponent an appropriate number of places to the right. In a two's complement system the right shift is accompanied with a sign-extension process. After the alignment process is complete, the mantissas are added. Next comes the problem of post normalization. Consider the sum of the largest legal mantissas:

$$\begin{array}{r} 0.1111 \\ +\,0.1111 \\ \hline 1.1110 \end{array} \tag{23}$$

This result is incorrect due to an "overflow" which changed the sign of the result. In this case, the correct result can be obtained by "sign-extending" the operands prior to the addition.

$$\begin{array}{r} 00.1111 \\ +\,00.1111 \\ \hline 01.1110 \end{array} \tag{24}$$

Note that this result requires a post normalization shift of one bit to the right with a corresponding increase in the exponent.

Now consider the case of adding relatively large negative mantissas using a 1-bit sign extension prior to adding:

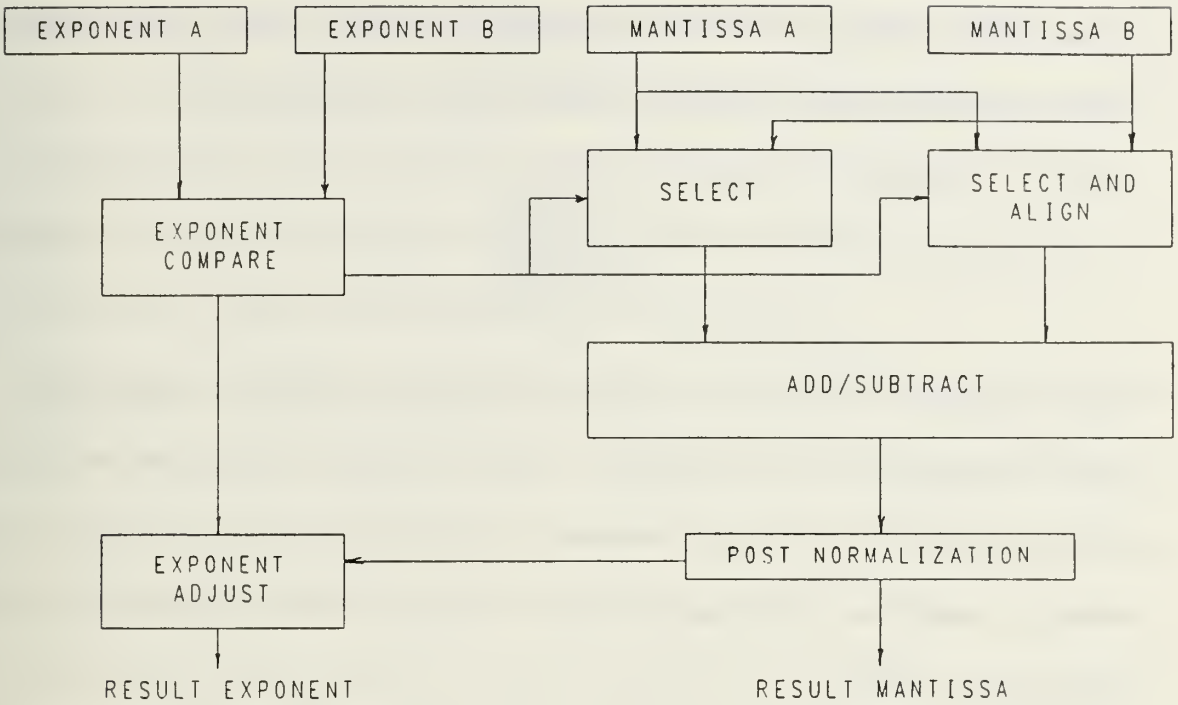


Figure 10. Block Diagram for Floating Point Addition.

$$\begin{array}{r} 11.0000 \\ +\,11.0000 \\ \hline 10.0000 \end{array} \tag{25}$$

This is the correct result, but it also requires a post normalization shift of one bit to the right, and a corresponding correction of the exponent.

The above two examples establish the limits of the magnitude increase with addition or subtraction. If one mantissa is shifted many places to the right in the alignment process, its magnitude compared with the unshifted mantissa is small. Therefore, the magnitude of the sum of the mantissas will be close to the value of the unshifted mantissa. However, if the magnitude of the mantissas are close

in value, but opposite in sign, the result may require a post normalization shift of many bits. Using a 5-bit two's complement example where the exponent compare results in a shift of a negative number one bit to the right:

$$\begin{array}{r} 00.1000 \\ +\ 11.10001 \\ \hline 00.00001 \end{array} \qquad (26)$$

Post normalization will require a shift of four places to the left. Thus the post normalization unit must be capable of a shift of at least one position to lesser significance, and shifts of many positions to higher significance. The post normalization must also adjust the exponent saved in the compare process for any shifts of the mantissa.

One last special case occurs when the resultant mantissa is precisely zero. For example:

$$\begin{array}{r} 00.1011 \\ +\ 11.0101 \\ \hline 00.0000 \end{array} \qquad (27)$$

Various methods have been devised to handle these "true" zero situations. The system developed in this thesis uses a mantissa of 0.000...0 *and* an exponent of 1000...0 to indicate a zero value. The post normalization process must also sense 0.000...0 in the mantissa and set the exponent to the required value.

E. OVERFLOW, UNDERFLOW, AND ROUNDING

Underflow occurs when an operation produces a result that is smaller in magnitude than the smallest representable non-zero magnitude. This can occur

in a multiplication process when two negative exponents are added and the result is more negative than 1000...0. This condition can be sensed by noting a sign change out of the exponent adder when two negative exponents are added. Underflow could also be caused by a post normalization in addition or multiplication. In the system developed in this thesis, when underflow is sensed, the exponent is set to 1000...0 and the mantissa is set to 0.000...0.

Overflow occurs when an operation produces a result that is larger in magnitude than the largest representable value. This can occur in multiplication when two positive exponents are added and the result is larger than 0111...1. This condition can be sensed by noting a sign change out of the exponent adder. Overflow could also be caused by a post normalization in addition. In the system developed in this thesis, when overflow is sensed, the exponent is set to 0111...1 and the mantissa is set to its largest magnitude while preserving its sign. A design may also incorporate an "error bit" which would be set if overflow occurs. As discussed earlier, because the maximum growth of data in a single stage of an FFT algorithm is known, certain hardware designs for specific systems may preclude the possibility of overflow. It is the responsibility of a system designer to use safeguards appropriate to the system.

The floating point operations of addition, subtraction, and multiplication increase the number of bits in the mantissa. There are many ways to deal with these extra bits. The simplest is truncation, which simply ignores these extra bits. Truncation throws away information, and results in a bias; the number to be stored is smaller than the true value [Ref. 16: p. 115].

Rounding, a commonly used method, adds half the value of the least significant bit position to the number before truncation. This results in a smaller average bias than the truncation method.

The technique of jamming was proposed by von Neumann to reduce overall error. This method "jams" a 1 into the least significant bit position of the result, regardless of the values of the extra bits. This method is as fast as truncation, but has a much smaller bias. Other techniques such as a round-to-zero scheme are available which use two bits of information to produce a result with zero bias. [Ref. 16: p. 116]

There is another consideration in the "extra bit" problem: how many of the extra bits should be retained in an addition alignment process? Consider a 7-bit mantissa (including sign bit) "aligned" five bits to the right for addition:

$$\begin{array}{r} 00.101101 \\ +\ 11.11111010101 \\ \hline 00.10101110101 \end{array} \qquad (28).$$

Reference 16: pp. 116-118 discusses this problem and shows that "at most three bits are needed beyond that required of the number system." These three bits may be needed in a round-to-zero zero bias technique. If simple rounding is used, only two extra bits need be saved.

The VLSI cells developed in this thesis use the simple rounding technique. The designs are easily modified for jamming or truncation. Other more complex techniques can be used with more modifications.

IV. HARDWARE

Registers and full adder cells form the basis of a pipelined adder design. Full adders are also used in many multiplier designs. Because most of the active chip surface will be composed of these two devices, many potential designs were evaluated. To keep comparisons on an equal basis, all layout simulations were made using "typical" CMOS specifications (see Appendix A for values), Vdd equal to +5.0 volts, and with lambda equal to 1.5 microns.

A. ADDER DESIGNS

Before designs for adders could be evaluated, a decision on whether to use look-ahead carry or ripple-carry adders was made. Figure 11 on page 34 [Ref. 20: p. 208] shows a gate-level design of a 4-bit look-ahead carry adder. Notice that several gates in the design have five inputs. Figure 12 on page 35 shows a gate-level design of a full adder that may be used in a ripple-carry circuit. The number of logic gate levels a signal passes through is sometimes referred to as the *depth* of the design. For a 4-bit ripple carry adder, the carry-out of the fourth stage is nine logic levels deep (one XOR, four AND's, and four OR's). The carry-out of the look-ahead carry adder is only three deep, but that can be deceiving; with CMOS logic, the maximum delay of a gate is proportional to the number of inputs in that gate [Ref. 11: p. 188]. CMOS delays are also sensitive to the number of gates driven by a particular gate (fan-out).

In order to more adequately evaluate the potential speed increase using look-ahead carry, test gates were designed and simulated using Spice. A significant

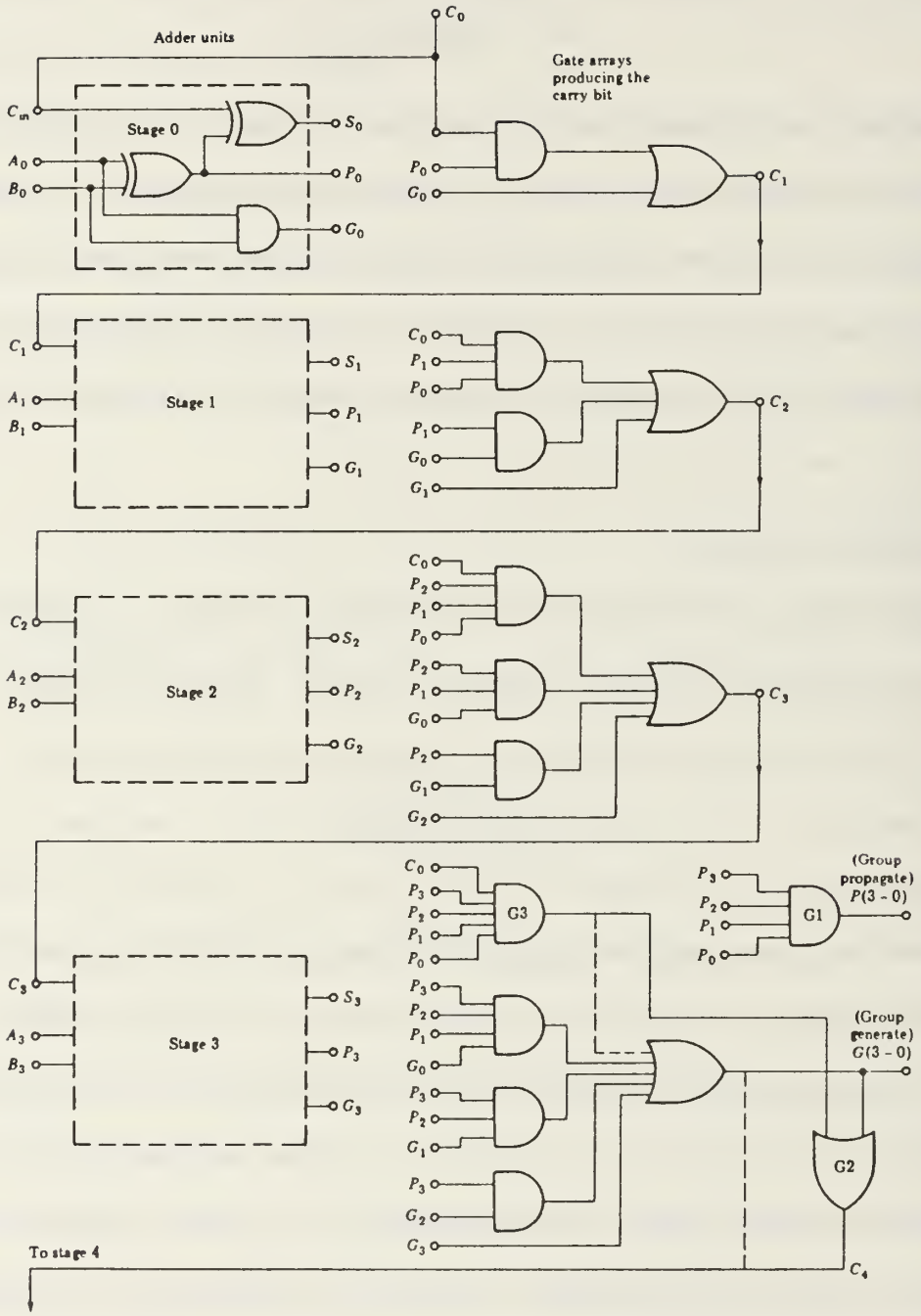


Figure 11. 4-bit look-ahead carry adder.

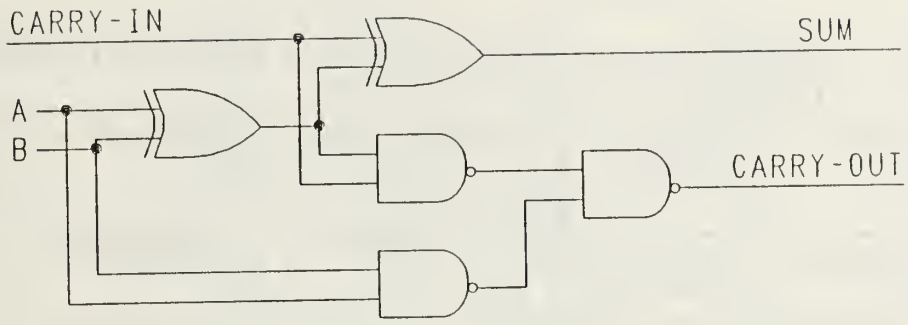


Figure 12. Full adder design.

speed advantage could not be obtained for the look-ahead device due to the gates with five inputs, and the larger fan-out of the look-ahead carry design. Reference 21: p. 155 also found that "the speed improvement [of a look-ahead carry adder] over a ripple adder is not significant for a small number of bits." In a pipelined arrangement only a relatively few number of additions will be done in any particular pipeline stage. Due to the necessity of producing simple, regular substructures with simple interfaces, and the limited speed improvement of a look-ahead carry adder in a pipelined design, ripple-carry adders were selected for use.

Many designs exist for implementing a full adder function. Table 3 on page 36 (from Ref. 12: p. 51) compares a conventional CMOS design with three other logic types. While it has the lowest power consumption, the conventional CMOS design is one of the slowest and has a much higher transistor count. If the transistor count could be reduced, the speed and size disadvantages of conventional CMOS logic might be minimized.

Table 3. COMPARISON OF PIPELINED FULL ADDER CELL IMPLEMENTATIONS

Full Adder Type	Maximum Clk. Rate	Average Power Dissipation at Max. f_{clk}	Power-Speed Ratio	Transistor Count
Conventional CMOS	100 MHz	0.500 mW	5.00 μW/MHz	32
Pseudo-NMOS	100 MHz	0.828 mW	8.28 μW/MHz	22
Standard N-P Domino	130 MHz	0.671 mW	5.16 μW/MHz	22
Quasi N-P Domino	165 MHz	0.962 mW	5.83 μW/MHz	23

Figure 13 shows a gate-level implementation of the exclusive-or (XOR) function. Using complementary logic, implementation of this design requires 16 transistors. Figure 14 on page 37 shows a transistor-level implementation of the XOR function using two transmission gates and an inverter as proposed by Ref. 11: p. 317. This design requires only six transistors. A full adder implementation of the gate-level design in Figure 12 using these transmission gate XOR's requires only 24 transistors. This is only one or two more transistors than other logic designs shown in Table 3.

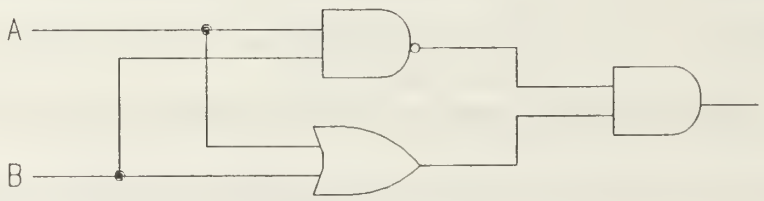


Figure 13. Exclusive-or function generation.

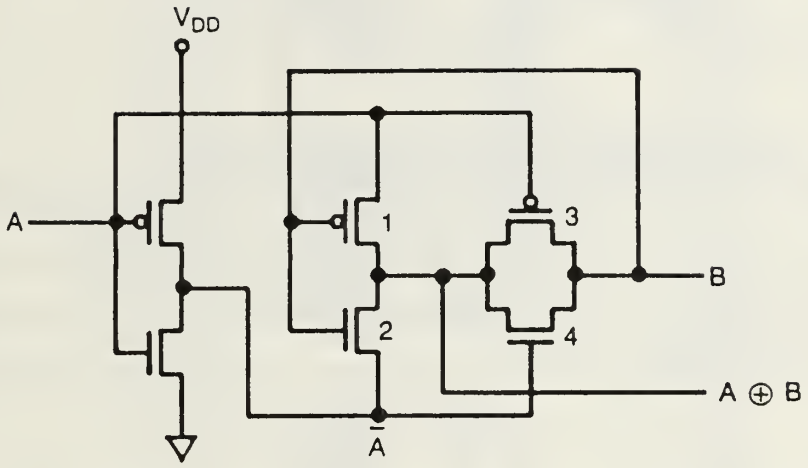


Figure 14. Exclusive-or function generation using transmission gates.

Many switch-level simulation programs will not satisfactorily simulate a CMOS XOR gate designed using transmission gates. Reference 22 describes the problem in detail and documents a computer program to modify input files and allow switch-level simulation of circuits using transmission gates.

A transmission gate XOR layout was simulated using Spice. Transistor widths were varied to achieve similar rise and fall times. The worst-case delay of the layout was 1.0 nanoseconds (ns). Simulation of an XOR gate using the NAND-OR-AND gates of Figure 13 on page 36 resulted in a maximum delay of 2.75 ns.

Due to the speed and size advantages of the transmission gate XOR, several full adder layouts using these XOR's were simulated. Transistor widths were varied to achieve approximately equal rise and fall times, and to reduce the carry-in to carry-out delay. Results of the simulation of the fastest full adder are

listed in Table 4 on page 38. Figure 15 on page 39 shows the layout design used in the simulation.

Table 4. FULL ADDER MAXIMUM DELAY TIMES.

EVENT:	A or B, or A and B change:	only CARRY-IN changes:
SUM-OUT delay	4.25 ns	2.75 ns
CARRY-OUT delay	4.5 ns	3.0 ns

The transmission gate has another useful feature; it produces the complement of one of the inputs. Figure 16 on page 40 shows that, if the complement is A is available, there is no need to invert A before the XOR gate; moving two connection points will accomplish the same function. Since two's complement subtraction can be performed by inverting the subtrahend and asserting the carry-in of the least significant bit of an adder, use of the transmission gate XOR will allow a subtraction cell design to have the same delays as a full adder. All subtraction cells developed in this thesis are modifications of full adder cells. Their delay times are the same as the full adder's delays (Table 4).

B. LATCH LAYOUTS

Pipelining requires devices to store logic states between stages. Various types of registers, flip-flops, and latches have been successfully used to perform this function. This thesis uses a compact pseudo 2-phase latch design proposed by Ref. 11: p. 206. Figure 17 on page 40 shows the latch design which requires eight transistors and four clock inputs. When $\phi 1$ is high (and $\overline{\phi 1}$ is low), the input, D, is inverted and node N is conditionally pulled high or low. When $\phi 1$ is

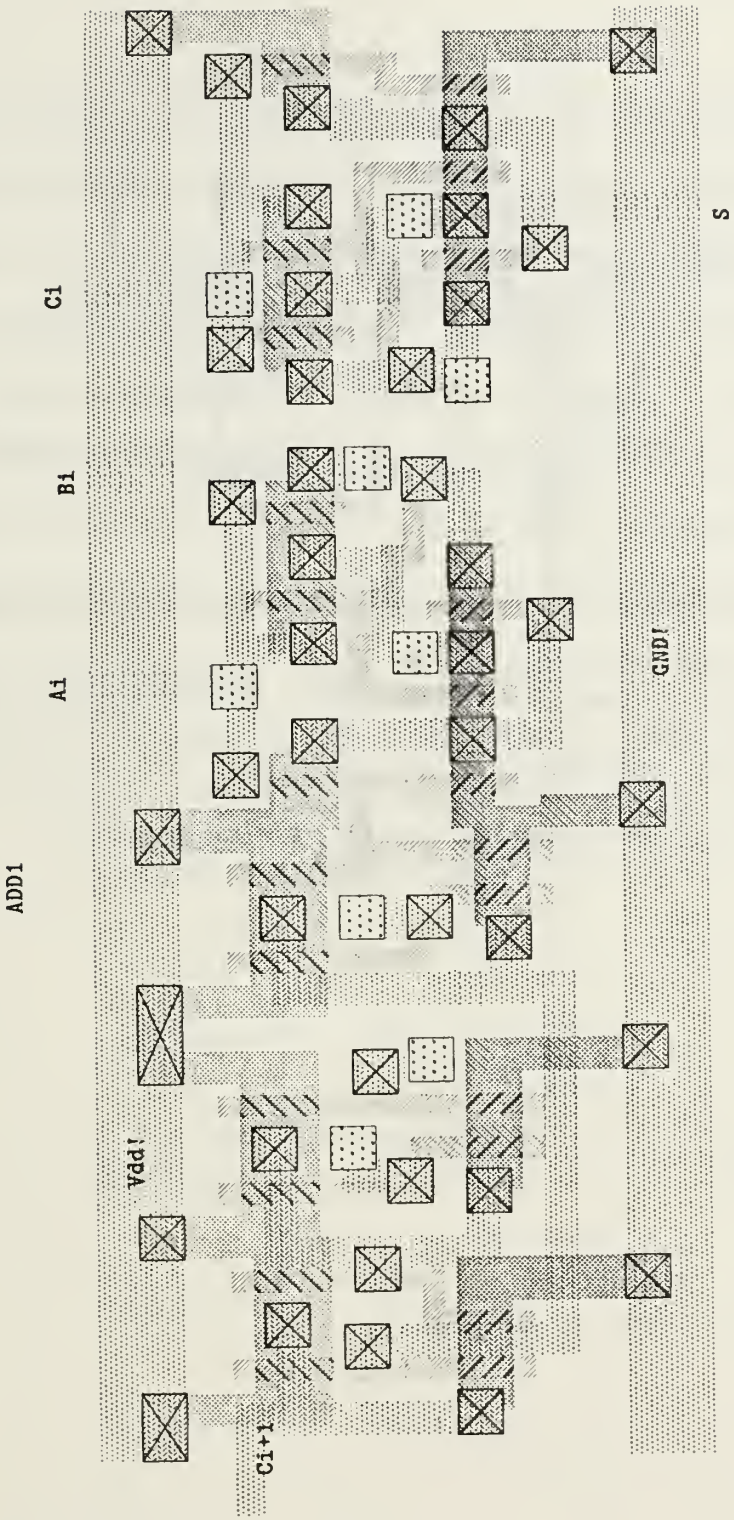


Figure 15. Full adder cell layout.

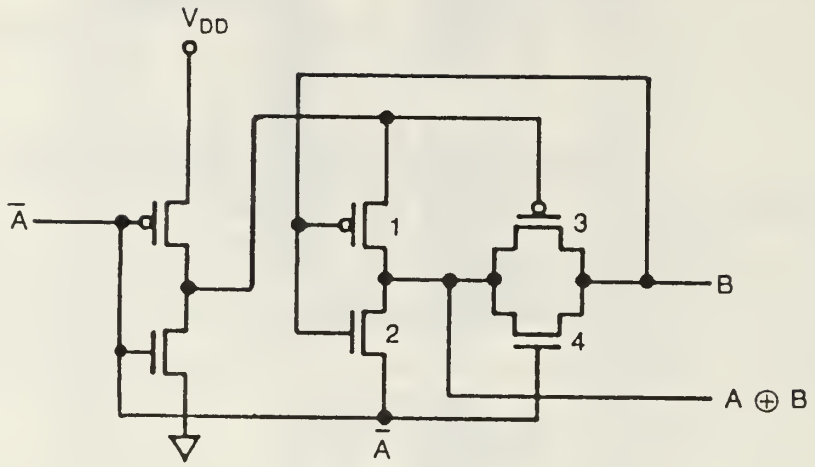


Figure 16. Exclusive-or function generation using complemented input.

low, the value on node N is stored. When $\phi 2$ goes high, the stored value is inverted and Q is conditionally pulled high or low. This design has several potential problems when implemented in high speed VLSI circuitry.

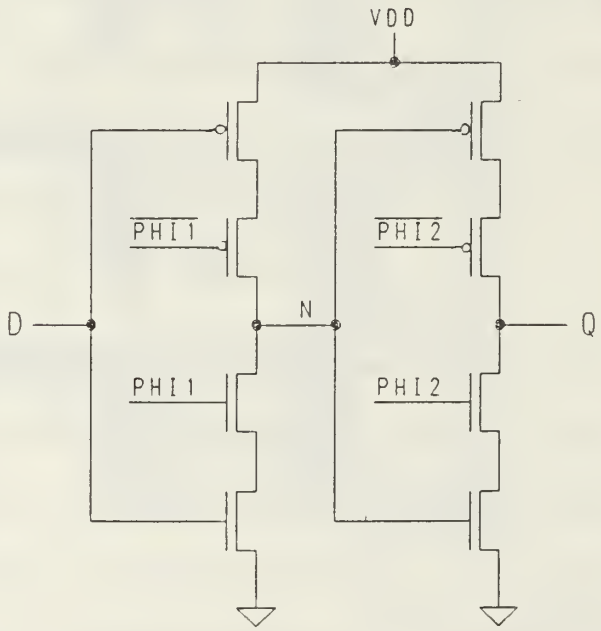


Figure 17. Pseudo 2-phase latch.

First, the design depends on the clock signals ($\phi 1$, $\overline{\phi 1}$, $\phi 2$, $\overline{\phi 2}$) to prevent the "latch" from becoming transparent. This would occur if $\phi 1$ and $\phi 2$ were high simultaneously. In a large design with many storage devices triggered off of clock lines, local clock skew could cause problems with timing and transparency.

Figure 18 shows the design modified with two inverters. The first inverter serves as a buffer for a single clock line. The second inverter produces a complement for use. Since $\phi 1$ and $\phi 2$ must never be high at the same time, and the latch also requires the complements of $\phi 1$ and $\phi 2$, the two signals can be used to supply the four clock lines of the latch. Simulation results show that the small amount of delay in level transition between $\phi 1$, and $\overline{\phi 1}$; and $\phi 2$, and $\overline{\phi 2}$ does not prevent proper functioning of the latch. This latch design was chosen due to its small size and its requirement of only one clock line.

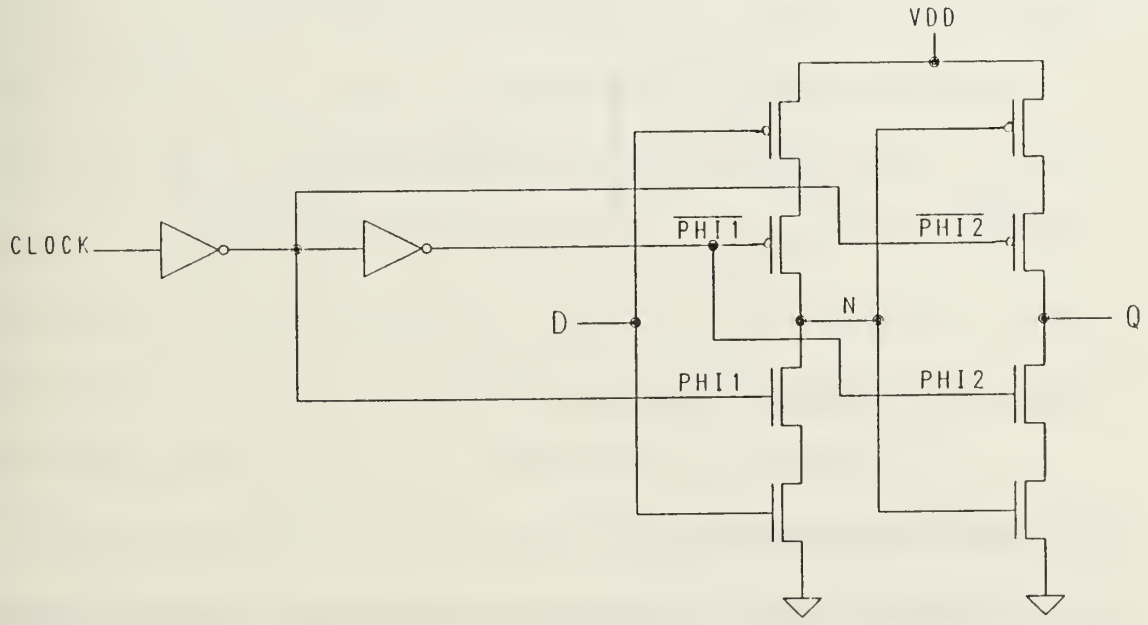


Figure 18. Modified clocks for pseudo 2-phase latch.

The latch design was refined with simulation in Spice to reduce the delay times and create approximately equal rise and fall times. Inverters were sized to produce rise and fall times of approximately 2 ns when driving the latches. The inverter transistor widths vary in different cells depending on the number of latches driven. To keep rise times fast and prevent skew problems, no inverter pair drives more than four latches in any cell design. The latch transistors were sized to produce a delay of 4 ns for *each* stage (input and output) of the latch. Figure 19 shows the clock pulse shape for the minimum allowable clock period, where $t_{logic_{max}}$ is defined as the longest logic delay in any stage of the pipeline. If a symmetric clock wave form is desired, the minimum clock period becomes

$$T_{C_{min}} = 2 \bullet (t_{logic_{max}} + 4\ ns). \tag{29}$$

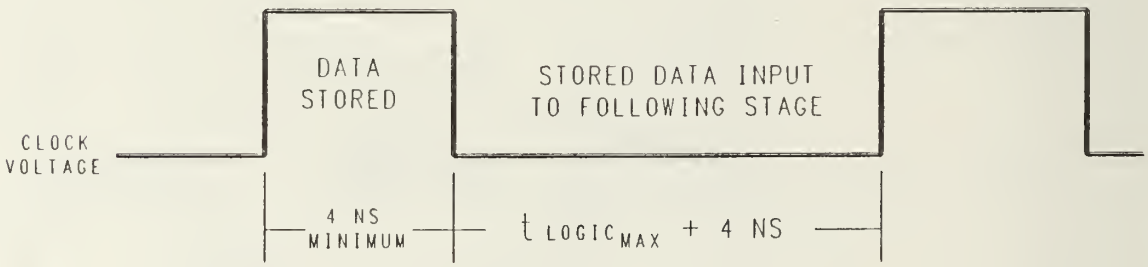


Figure 19. Clock pulse requirements.

C. MULTIPLIER DESIGNS

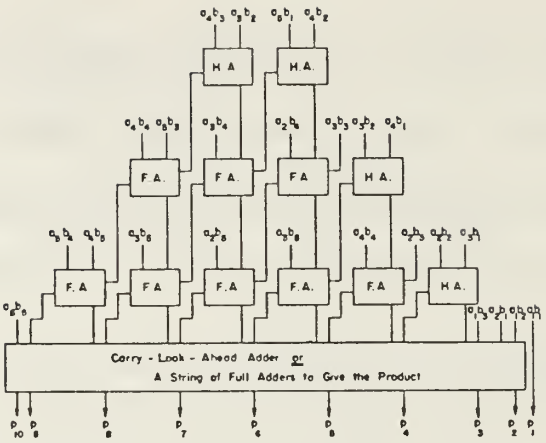
The following formula illustrates the operations required to multiply two 4-bit binary numbers.

$$
\begin{array}{ccccccc}
 & & & X_3 & X_2 & X_1 & X_0 \\
 & & \times & Y_3 & Y_2 & Y_1 & Y_0 \\
\hline
 & & & x_0y_3 & x_0y_2 & x_0y_1 & x_0y_0 \\
 & & x_1y_3 & x_1y_2 & x_1y_1 & x_1y_0 & \\
 & x_2y_3 & x_2y_2 & x_2y_1 & x_2y_0 & & \\
x_3y_3 & x_3y_2 & x_3y_1 & x_3y_0 & & & \\
\hline
P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
\qquad (30)
$$

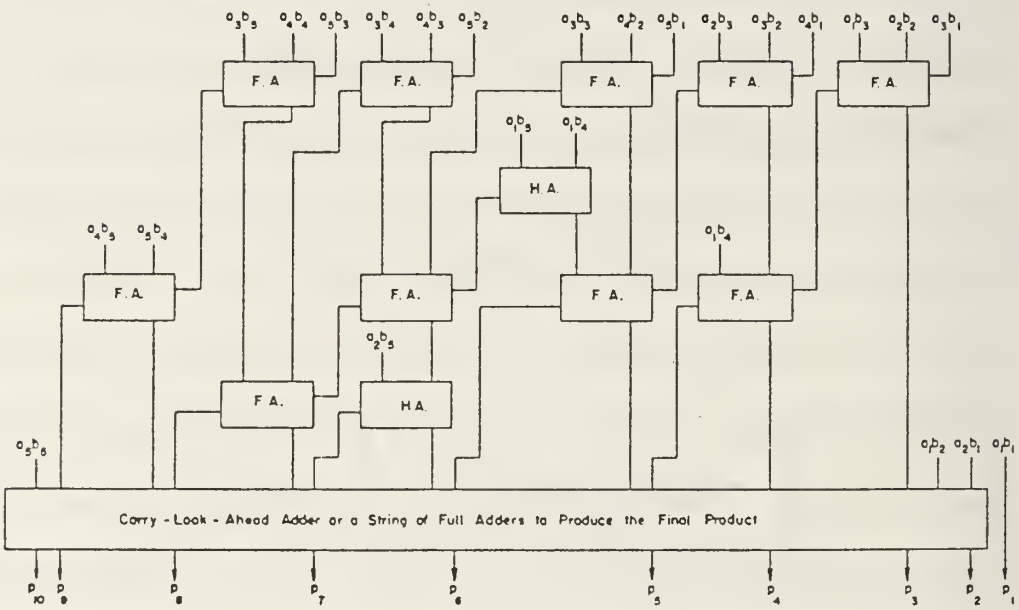
The partial products (x_iy_i's) are formed by the AND function. The columns of partial products are then added to produce the product.

Reference 21 describes and analyzes three design schemes for adding the columns of partial products. Two of the schemes (Figure 20 on page 44) strive to minimize the time required for a multiplication by reducing the number of logic levels in a design. These level reduction methods increase wiring complexity and do not produce regular rectangular-shaped VLSI layouts. Pipelining allows a system to have a high throughput rate which is not dependent on the number of logic levels in a design; therefore, minimizing the number of logic levels is not of prime importance. The third design type, a carry-save scheme, is shown in Figure 21 on page 45 (From Ref. 21: p. 156). This array employs straightforward full adders in an architecture in which a modular structure and repetitive layout can be maintained. Due to this regularity and modularity, a carry-save array design was chosen for the multiplier layouts in this thesis.

The array multiplier can be "reshaped" slightly to form a rectangular array as shown in Figure 22 on page 46 (from Ref. 11: p. 345). Since other devices, such as adders, used in the thesis designs have rectangular layouts, the use of a



Block diagram of a 5×5 multiplier using Dadda's scheme.



Block diagram of a 5×5 multiplier using Wallace's scheme.

Figure 20. Wallace tree and Dadda multipliers.

rectangular shape in the multipliers allows for more efficient overall usage of VLSI chip area.

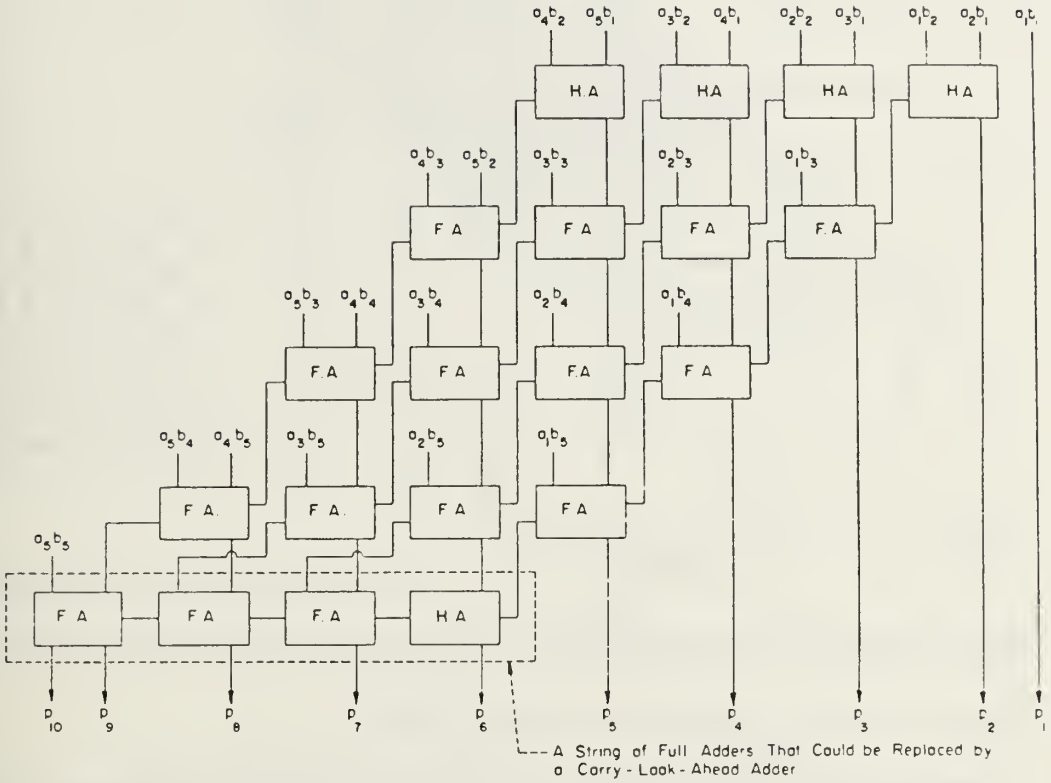


Figure 21. Block diagram of a 5 by 5 multiplier using carry-save technique.

The functions of cells in the array are shown in Figure 23 on page 47. The partial products ($x_i y_j$'s) are formed in the individual multiplier cells by the AND gate. This array design and layout keeps communication distances at a minimum; once the operands are distributed to the multiplier cells, no cell communicates to any cell other than its nearest neighbor. This eliminates many of the delay times necessary in other multiplier types and makes the design simpler to implement. This design can also be easily modified for various length operands simply by changing the number of rows and columns. This parallel array also

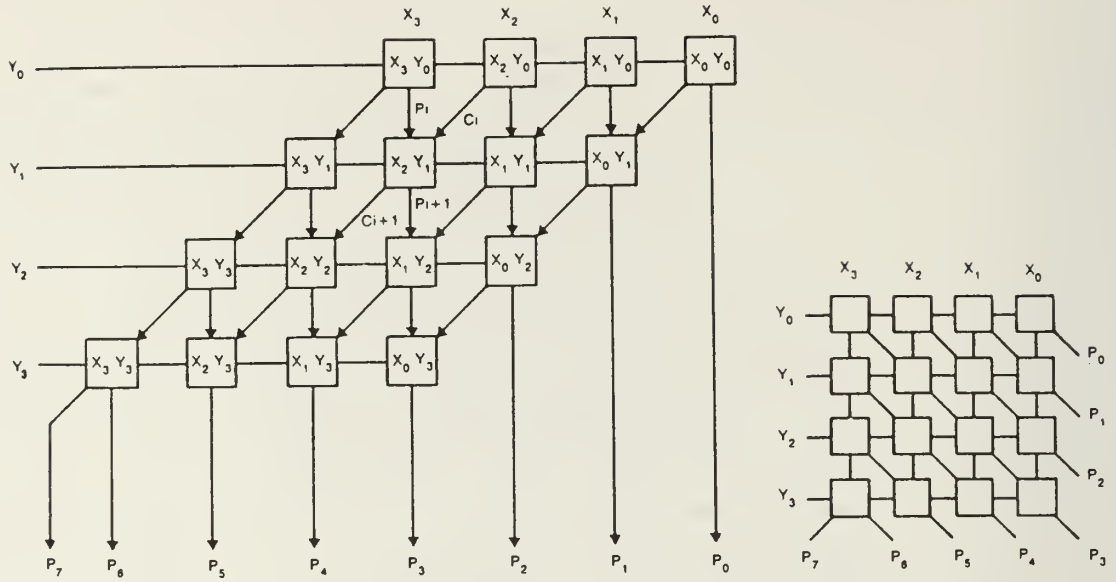


Figure 22. Parallel multiplier array.

appears well-suited for pipelining; latches can be placed between "stages" as necessary to produce acceptable throughput rates.

The multiplier cells used in this thesis perform the functions shown in Figure 23 on page 47. Because a CMOS AND gate is a NAND gate followed by an inverter, the multiplier cells use NAND gates vice the AND gates shown. Since the transmission gate XOR design can be arranged to accept an input or its complement, this eliminates an inverter in the critical timing path. A gate level design for the multiplier cell is shown in Figure 24 on page 48. Delay times obtained by simulation using Spice are given in Table 5 on page 47. Figure 25 on page 49 shows the layout of the multiplier cell used in the simulation.

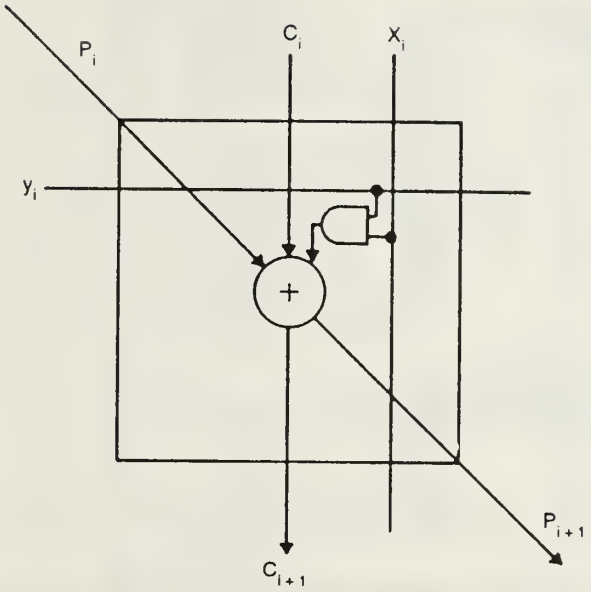


Figure 23. Parallel multiplier cell

Table 5. MULTIPLIER CELL MAXIMUM DELAY TIMES.

EVENT:	X_i and Y_i change	P_i changes	C_i changes
Delay for P_{i+1}	5.75 ns	4.25 ns	2.75 ns
Delay for C_{i+1}	6.0 ns	4.25 ns	3.0 ns

D. OTHER CELL FEATURES.

In addition to the duties performed by the addition, subtraction, and multiplication cells, many other functions such as rounding, mantissa alignment, and normalization are necessary. Brief descriptions of these additional functions are provided here. The appendices contain more detailed cell composition and layout information.

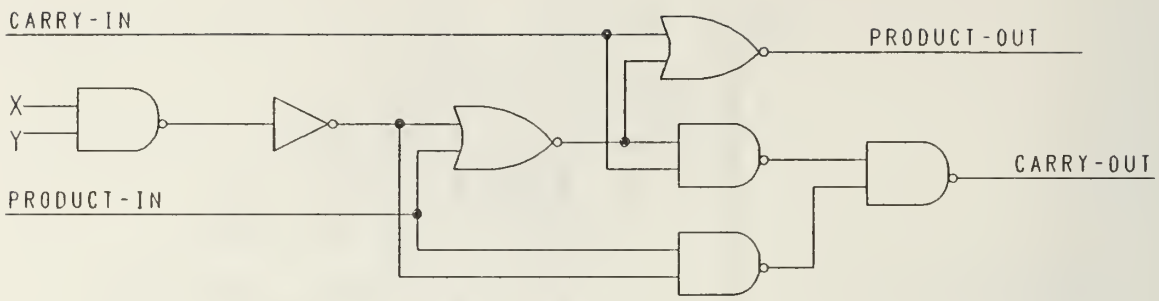


Figure 24. Multiplier cell design.

A gate-level logic diagram for two's complement conversion, required when changing between addition and multiplication, was shown in Figure 9 on page 20. Because a CMOS OR gate is composed of a NOR gate followed by an inverter, the conversion speed can be increased slightly by modifying the design. Figure 26 on page 50 shows the modified design; Table 6 lists the maximum delay times of the modified design.

Table 6. TWO S COMPLEMENT CONVERSION DELAY TIMES.

A_{TC0}	A_{TC1}	A_{TC2}	A_{TCi}
0.0 ns	1.0 ns	3.0 ns	$[(i-2) \cdot 2 + 3]$ ns

As shown earlier, in the multiplication process, mantissa alignment may be required. This shift requires subtracting 1 from the sum of the operand exponents. Many systems follow the alignment process with the exponent adjustment process. In a hardware pipelined system, if an operation might be required, the pipeline stages necessary to perform the operation, and circuitry to determine

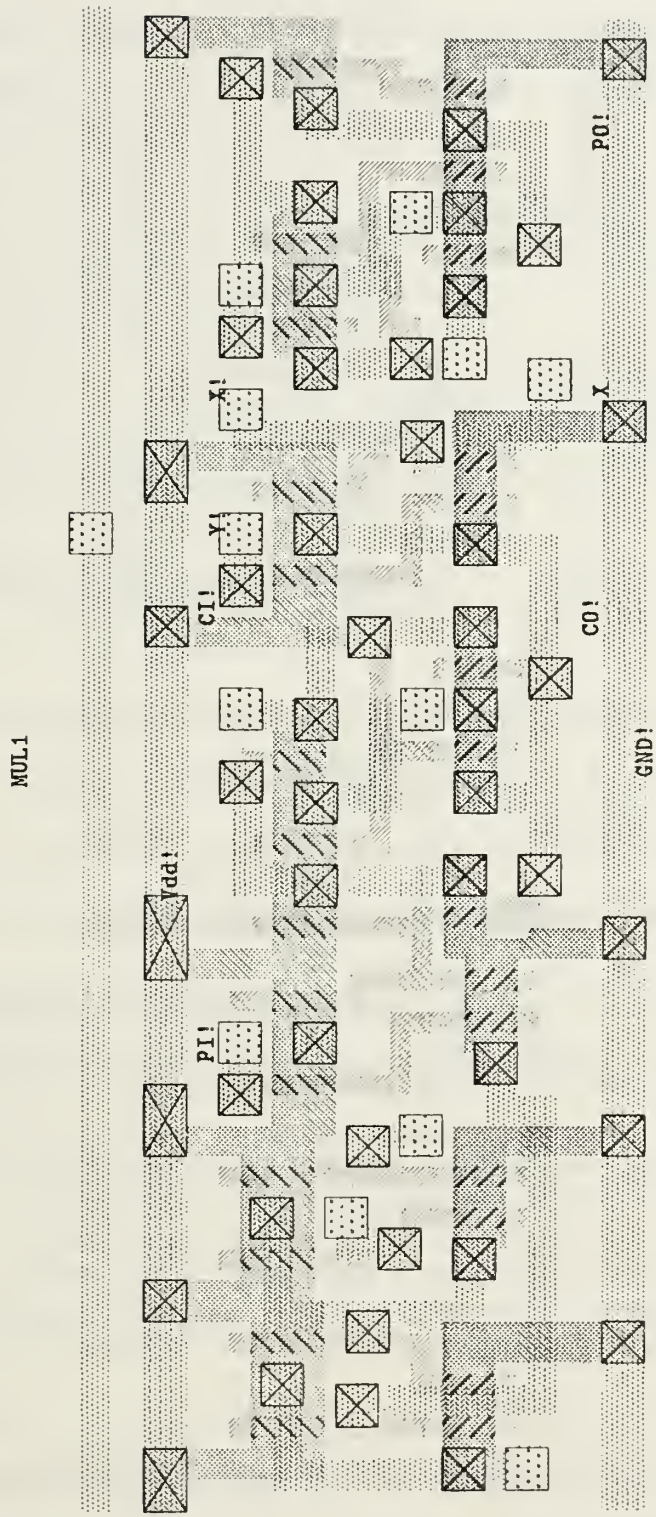


Figure 25. Multiplier cell layout

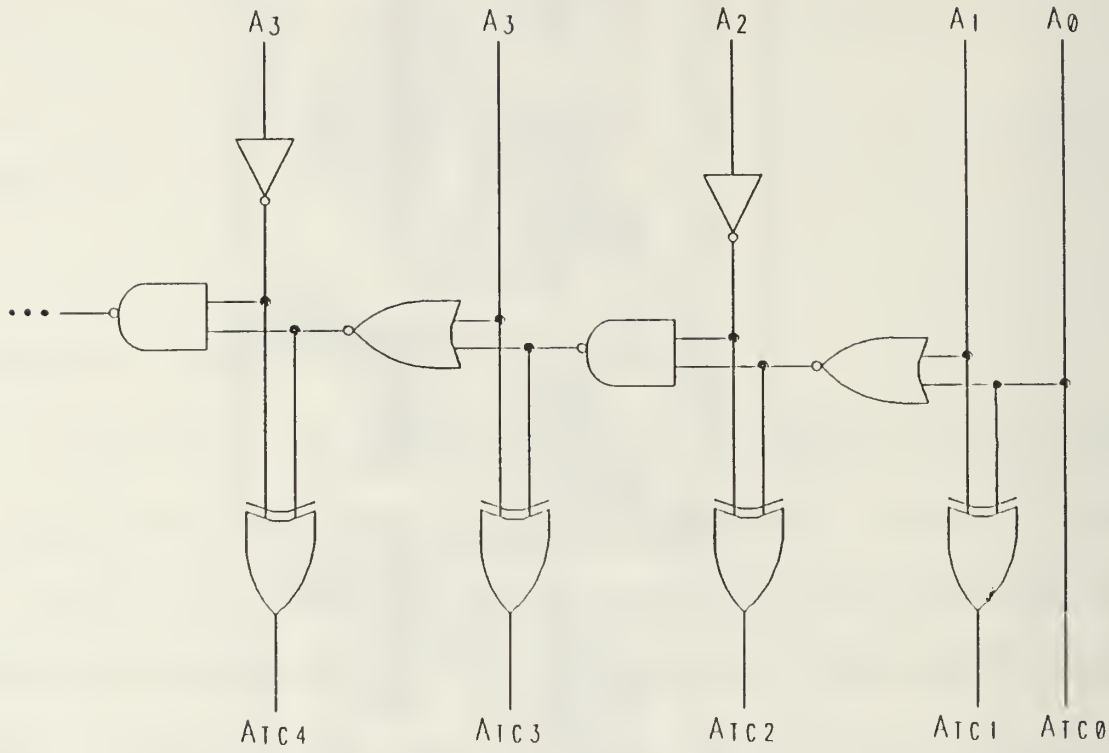


Figure 26. Two's complement converter (modified design).

whether the process is necessary, and route signals is necessary. Waiting until after the multiplication is complete to start the subtract 1 process is time consuming; the subtraction process must then ripple through more stages of a pipeline. Time and chip area can be saved by precomputing the value for the exponent if alignment is necessary, and then "selecting" the appropriate exponent value.

Figure 27 on page 51 shows a design implemented in the multiplication exponent addition layout to perform this computation process. This design is used to modify all bits except the least significant bit, which is always the inverse of

the unmodified bit. Note that both the exponent and the modified exponent are retained for possible use.

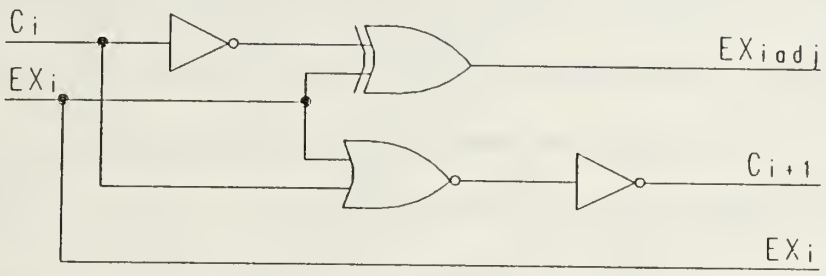


Figure 27. Design to subtract 1 from exponent.

There are two other possible adjustments to the exponent in the multiplication process. Underflow requires setting the exponent to 1000...0; overflow requires setting the exponent to 0111...1. Figure 28 on page 52 shows the transmission gate network used to select the appropriate exponent at the end of the of the multiplication process.

In the multiplication process, precomputation can also be used to speed up the mantissa rounding process. Since there are only two possible locations of the radix point, the rounding process can be performed from both starting points, and both results stored until the location of the radix point is known and the appropriate value is selected. When overflow occurs, the mantissa is set to either 1.000...0 or 0.111...1 (depending on its sign). When underflow occurs, the mantissa is set to 0.000...0. A shift network very similar to the one used for exponent selection is used to select the correct mantissa value from the four possible choices.

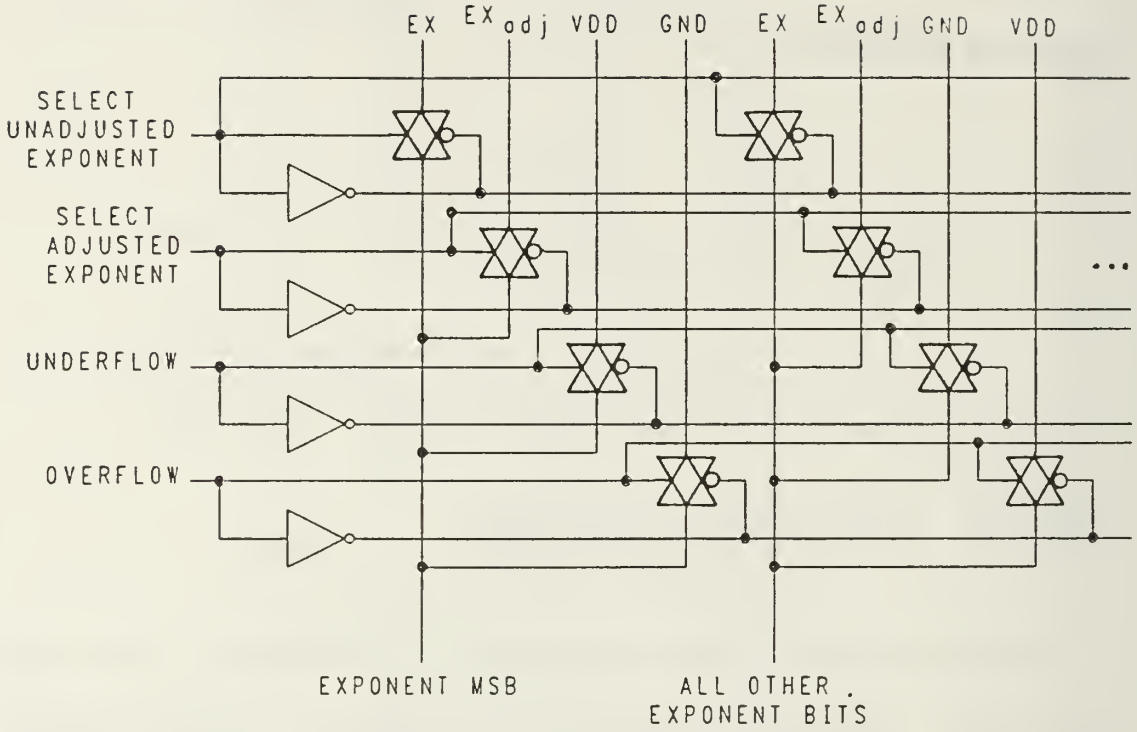


Figure 28. Exponent selection network.

Figure 29 shows the gate-level diagram of the design used in the rounding process. The carry-in of the least significant bit is set with the value of the product bit one position to the right of the least significant bit.

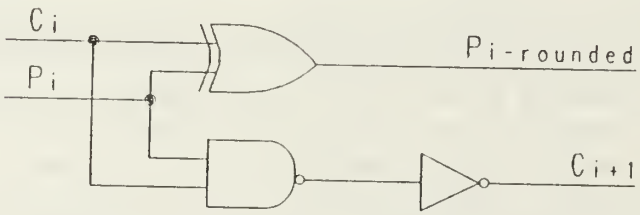


Figure 29. Rounding design.

Zero sensing is also required in the multiplication process. If one of the multiplication operands is a true zero, and the exponent was not set to 1000...0 at the end of the process, the mantissa alignment in the next floating point

addition could be erroneous. (The larger magnitude mantissa might be shifted.) With the floating point system proposed in this thesis, checking the two most significant bits (excluding the sign bit) for a 1 is all that is necessary to check for the true zero condition. This can be accomplished with a single NOR gate.

The first step in a floating point addition or subtraction is to subtract the exponents. The difference is the amount of mantissa alignment necessary for the smaller magnitude operand. The shift network used to align the mantissas is capable of shifts up to 31 bit positions to the right. This shift network uses the five least significant bits (LSB's) of the difference to determine the number of places shifted. With the two's complement number system used for exponents, the shift value out of the exponent subtraction cells could be positive or negative. The "sign" bit of the difference is used to determine which exponent to store and which mantissa to align. Since the mantissa shift network requires positive bit values to function properly, the exponent subtraction cells also concurrently take the two's complement of the five LSB's. Both sets of LSB's are stored until the sign of the operation is known, and the appropriate selection of control bits for the shift network is made.

The mantissa alignment of the present design is limited to 31 bits. Modifications to the design is possible to allow any magnitude shift. If the exponent subtraction process indicates that more than 31 bit places of shift is necessary, the shift is reduced to 31 places. For positive values of the difference in exponents, this condition is indicated by the presence of a 1 in any bit position more significant than the five LSB's. It is sensed with a string of OR gates as shown in Figure 30 on page 54. Similarly, when the difference in exponents is negative, a

1 in any bit position more significant than the five LSB's of the two's complement of the difference indicates a shift of more than 31 places. If a shift of more than 31 places is called for, 1's are loaded into the shift network control to obtain the 31 place shift. Figure 31 on page 55 shows a gate-level diagram of the network used to determine whether to store the "A" exponent and shift the "B" exponent or vice versa. In this design, B is subtracted from A; the difference is labeled D. A negative number indicates that B's exponent was larger and exponent B is stored and the mantissa of A is shifted in the alignment process. The signals POS1 and TC1 are produced by the chains of OR gates that sense 1's in bit positions more significant than the 5 LSB's. The signal LOAD_1S is OR'ed with each of the five LSB's of the selected difference. If LOAD_1S is high, all five shift control bits are high. If LOAD_1S is low, the OR gates pass the shift bits through unaltered.

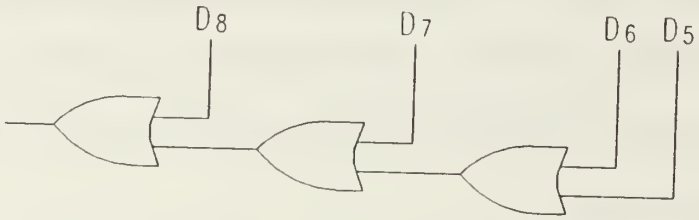


Figure 30. Ones' sensing design.

The mantissa alignment network functions in two stages. First, the most significant control bit shifts the mantissa either 0 or 16 places. Figure 32 on page 56 shows one cell of the shift network. When the control bit, C_4, is a 0, the i_{th} bit is selected in the i_{th} stage, resulting in a shift of 0 places. When the control bit is a 1, the $i + 16th$ bit is selected resulting in a shift of 16 places.

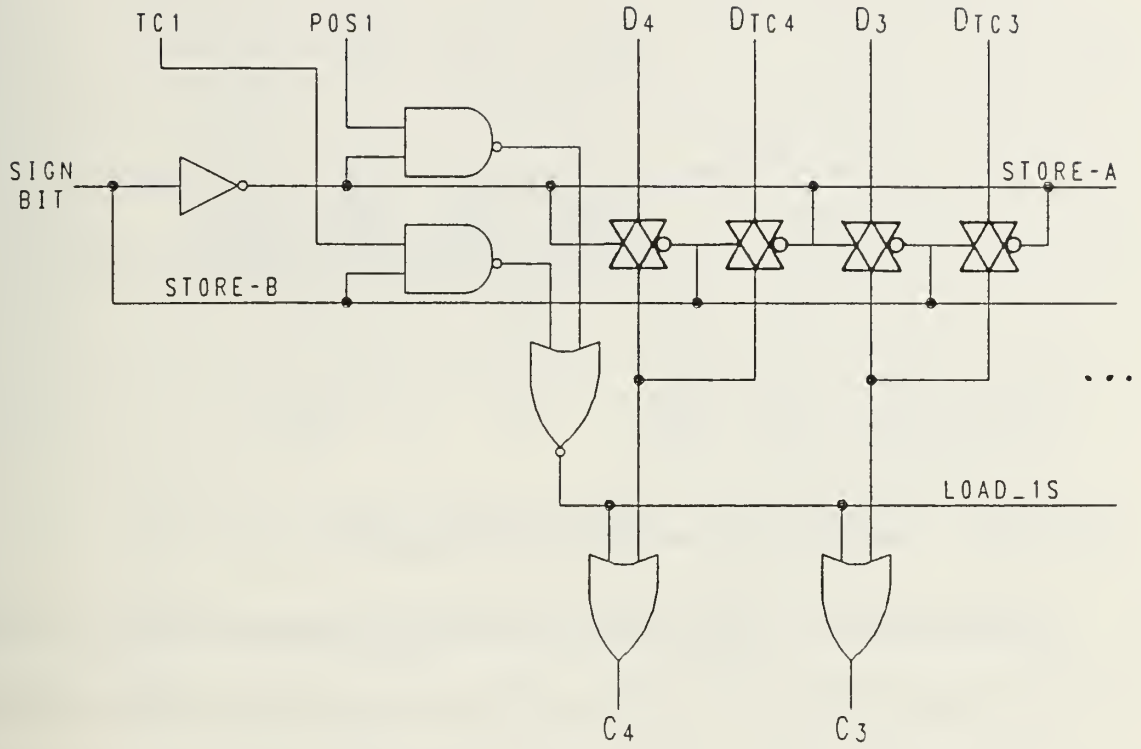


Figure 31. Shift network control selection design.

The second-stage of the shift network uses the four least significant control bits, C_3 through C_0, to shift from 0 to 15 places. Figure 33 on page 57 shows a cell and the decoders necessary to control the shift. The inverters are used to "buffer" the long signal line runs and speed up overall operation. One of the upper four transmission gates select a shift of 0, 4, 8, or 12 places while the lower transmission gates select shifts between 0 and 3 places. The combination results in shifts of 0 to 15 places. The magnitude of the shift is determined by the value of the control bits.

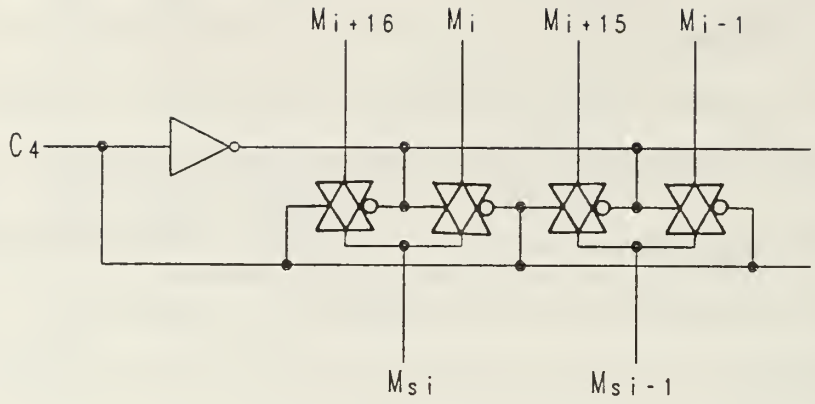


Figure 32. First-stage mantissa alignment shift network.

The combination of both stages of the mantissa alignment allows for shifts of 0 to 31 places. Since at most two extra bits need to be saved in the alignment process, the 31-bit shift capability allows for mantissa lengths (including the "sign" bit) of up to 30. (A 30-bit mantissa shifted 31 places to the right and "sign filled" produces 32 bits, all equal to the sign bit.) If more than 30-bit length mantissas are desired, the system must be modified.

Because a two's complement number system is used, any mantissa that is shifted to the right requires filling in the most significant bits with the value of the "sign" bit. (While the most significant bit of a two's complement number is not technically a sign bit, its value determines whether the operand represented is positive or negative. This leading bit is sometimes loosely referred to as a sign bit in this thesis for simplicity of reference.) Due to this sign filling process, the sign bit has a fan-out of 16 bit lines in the first-stage shifter. A designer must buffer this sign bit as necessary to ensure proper operation of the system.

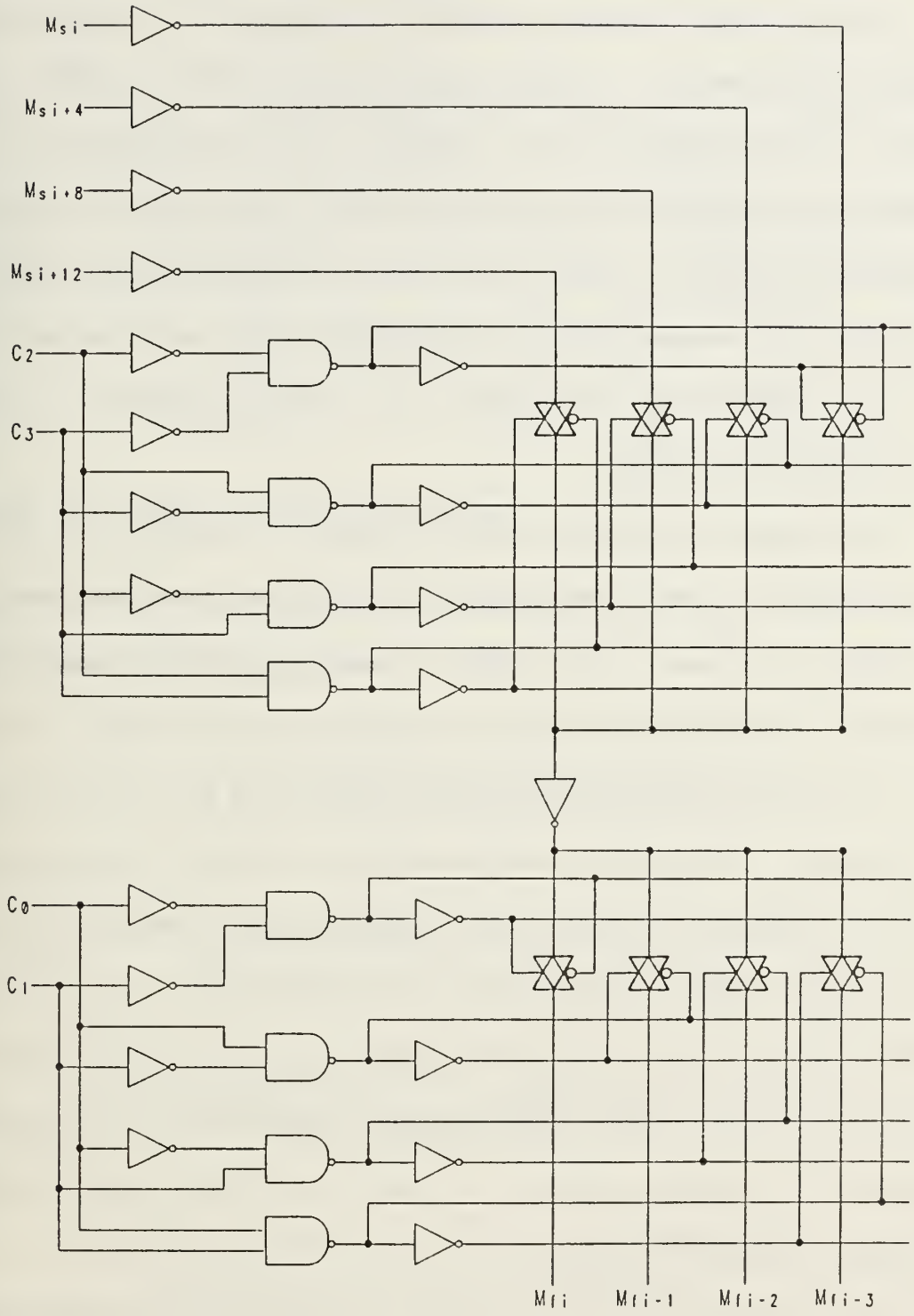


Figure 33. Second-stage shift network.

After the aligned mantissas are added (or subtracted), a post normalization procedure is necessary. As discussed earlier, this normalization could be from 1 place to the right to many places to the left. The maximum left shift is N-1, where N is the length of the mantissa (including the sign bit). For a 30-bit long mantissa, provisions for shifts of from 1 bit to the right to 29 bits to the left are necessary. This normalization can be accomplished with a shift network that is a "mirror image" of the one described for a mantissa alignment if the proper control signals are available.

To determine the required shift, the location of the left-most 01 or 10 bit pattern must be sensed. This location can then be used to code the control lines and adjust the exponent value. There are several possible ways to sense the bit patterns. This thesis uses a "ripple" arrangement shown in Figure 34 on page 59. The EXCLUSIVE-OR gates output is high if a 10 or 01 bit pattern exists. The string of NOR gates and inverters sense and "remember" the existence of a 1 out of an EXCLUSIVE-OR of more significant bits. Only one NAND gate (and possibly none) will output a 0 for any possible pattern. The output of the NAND gates can be sent to an encoder design to activate the control lines for the post normalization network. This design has a drawback in the time required to perform the operation; sufficient time must be provided to allow a signal to propagate through the OR gates.

An encoder design is shown in Figure 35 on page 60. This design OR's 16 inverted inputs to produce each of the five control bits used in the post normalization process. The inputs to the initial NAND gates are the NAND gate

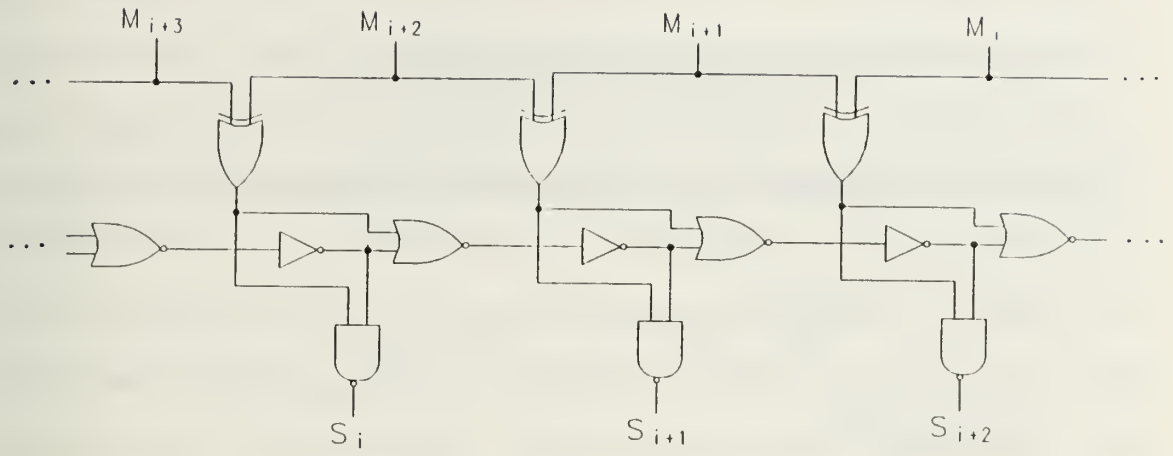


Figure 34. Radix point location sense design.

outputs from the bit pattern sensing arrangement of Figure 34 on page 59. The inputs to each control encoder are:

$$C_0 = S_1 + S_3 + S_5 + S_7 + S_9 + S_{11} + S_{13} + S_{15} + S_{17} + S_{19} + S_{21} + S_{23} + S_{25} + S_{27} + S_{29} + S_{31}, \tag{31}$$

$$C_1 = S_2 + S_3 + S_6 + S_7 + S_{10} + S_{11} + S_{14} + S_{15} + S_{18} + S_{19} + S_{22} + S_{23} + S_{26} + S_{27} + S_{30} + S_{31}, \tag{32}$$

$$C_2 = S_4 + S_5 + S_6 + S_7 + S_{12} + S_{13} + S_{14} + S_{15} + S_{20} + S_{21} + S_{22} + S_{23} + S_{28} + S_{29} + S_{30} + S_{31}, \tag{33}$$

$$C_3 = S_8 + S_9 + S_{10} + S_{11} + S_{12} + S_{13} + S_{14} + S_{15} + S_{24} + S_{25} + S_{26} + S_{27} + S_{28} + S_{29} + S_{30} + S_{31}, \tag{34}$$

$$C_4 = S_{16} + S_{17} + S_{18} + S_{19} + S_{20} + S_{21} + S_{22} + S_{23} + S_{24} + S_{25} + S_{26} + S_{27} + S_{28} + S_{29} + S_{30} + S_{31}, \tag{35}$$

where S_0 is the NAND gate output of the EXCLUSIVE-OR between the two most significant bits of the unnormalized mantissa. (The + sign represents an OR operation.) Grouping of the same terms used in different encoders might result in some reduction in gate numbers while increasing the wiring complexity.

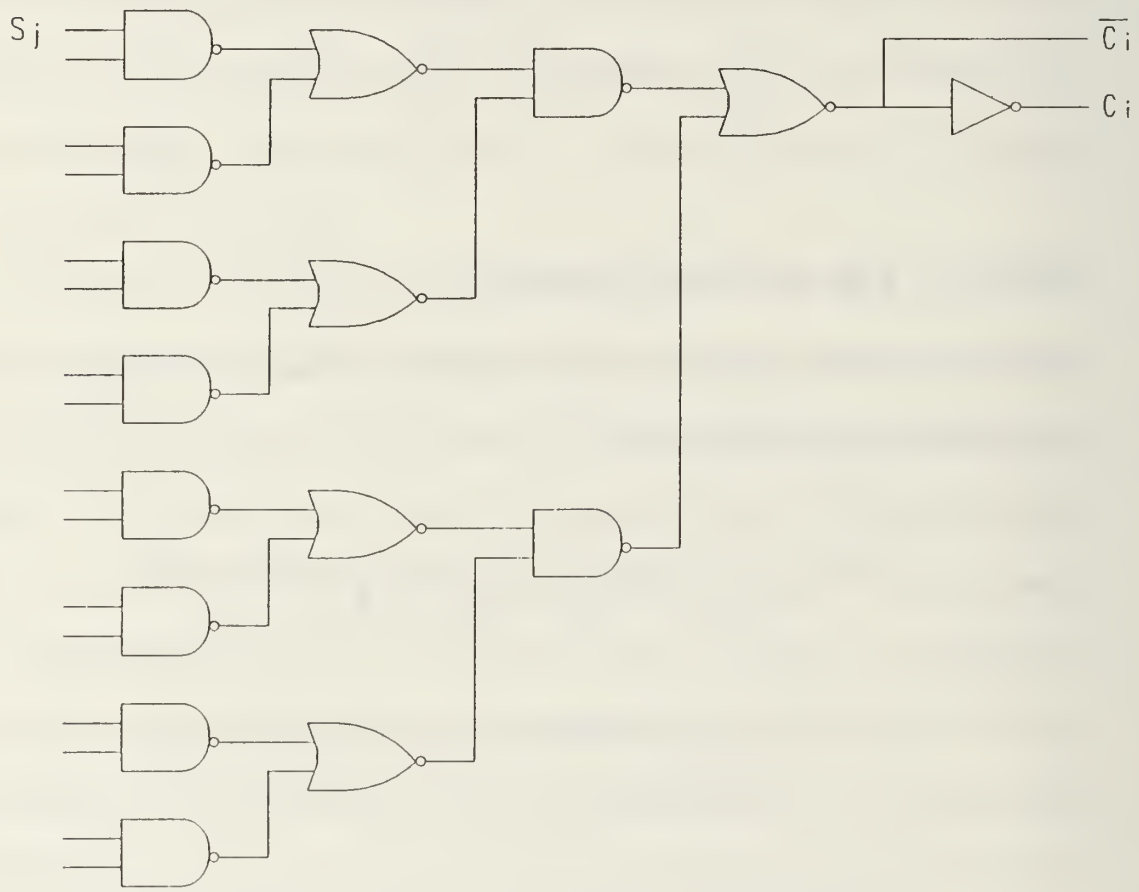


Figure 35. Encoder design for post normalization control lines.

V. DESIGN COMPARISONS

In order to demonstrate pipelining techniques and compare the size and speed capabilities and layout area requirements of multipliers and adders developed in this thesis, several example layouts are shown in the appendix. As a design example, and to keep comparisons consistent, all layouts use no more than three multiplier cells or five full adder or subtracter cells in a single pipeline stage. The maximum logic delays, $t_{logic_{\max}}$, for three multiplier or five full adder cells with $\lambda = 1.5$ microns are

$$t_{logic-3mul_{\max}} = 6.0 + 2 \bullet 4.25 \ ns = 14.5 \ ns \tag{36}$$

$$t_{logic-5add_{\max}} = 4.5 + 4 \bullet 3.0 \ ns = 16.5 \ ns \tag{37}$$

Note that the logic delay for the five full adder cells is the critical timing path. The logic delays in pipeline stages with other logic functions were kept below the five full adder maximum delay times to make the addition cells the critical timing path in determining the maximum clock rate. Table 7 on page 62 lists the predicted maximum pipeline throughput possible using that critical timing path for various feature sizes.

For comparison purposes, the layout size requirements of the example systems were determined. 16-bit mantissa (including sign bit), 12-bit exponent floating point devices are compared with 16-bit and 29-bit fixed point devices. The 29-bit fixed point example was chosen due to its association with a 16-bit

Table 7. PREDICTED MAXIMUM PIPELINE THROUGHPUT.

FEATURE SIZE:	3.0 microns	2.0 microns	1.2 microns
Asymmetric clock	40.8 MHz	61.2 MHz	102.0 Mhz
Symmetric clock	24.4 MHz	36.6 MHz	61.0 MHz

floating point format with a 1024-point FFT. A 1024-point FFT requires ten radix-2 FFT butterfly stages. Using the maximum growth factor per FFT stage of 2.414 (from formula (7)), the maximum growth for the entire FFT is

$$2.414^{10} = 6720. \tag{38}$$

Noting that $2^{13} = 8192$, 13 extra bits should be sufficient to prevent overflow and eliminate the need to scale results between stages. Table 8 on page 63 shows the VLSI chip area required for various devices. The table listings do not include areas for pads, line routing between devices, etc., so the final chip size requirements will be significantly larger than that shown.

Individual cell layouts were made with the goal of producing function designs that are composed of individual cells that abut other cells and require no additional connections or routings except at the periphery. Figures 36, 37, and 38 show the arrangement of cells necessary to produce a 16-bit mantissa multiply, a 12-bit exponent addition (used in a floating point multiply process), and a 12-bit exponent subtraction (used in the exponent compare process of a floating point addition or subtraction). Due to the complicated flow paths in a floating point addition process, the number of cell designs required to perform the function with

no extra internal routings would be prohibitive. Use of the standard cell designs to perform a floating point addition presently requires numerous connections and wire routings between various functional blocks.

The above examples are for comparison only. The cell system described in this thesis is capable of producing designs with any number of logic cells in a pipeline stage. The theoretical maximum pipeline throughput of a multiplier with pipeline latches between each multiplier cell using 1.2 micron feature size technology is 178.5 MHz. This extreme speed is probably not practically achievable; I/O interfaces will normally limit overall system throughput rates.

Table 8. VLSI CHIP AREA USED IN CUSTOM LAYOUTS

FEATURE SIZE	3.0 microns	2.0 microns	1.2 microns
16-bit fixed point multiplication	12.30 mm^2	5.47 mm^2	2.68 mm^2
29-bit fixed point multiplication	43.15 mm^2	19.0 mm^2	9.40 mm^2
16-bit mantissa, 12-bit exponent multiplication	16.6 mm^2	7.38 mm^2	3.62 mm^2
16-bit fixed point addition	2.06 mm^2	0.917 mm^2	0.449 mm^2
29-bit fixed point addition	4.99 mm^2	2.22 mm^2	1.09 mm^2
16-bit mantissa, 12-bit exponent addition	14.27 mm^2	6.43 mm^2	3.11 mm^2
16-bit fixed point radix-2 FFT	0.616 cm^2	0.274 cm^2	0.134 cm^2
29-bit fixed point radix-2 FFT	2.03 cm^2	0.893 cm^2	0.438 cm^2
floating point radix-2 FFT	1.52 cm^2	0.676 cm^2	0.330 cm^2

Reference 23, a thesis completed at the Naval Postgraduate School, documents the chip area and performance specifications of a pipelined multiplier design produced using the Genesil Silicon Compiler and its standard library cells.

Figure 36. Cell structure for a 16-bit mantissa multiply.

Figure 37. Cell structure for a 12-bit exponent subtraction.

Figure 38. Cell structure for a 12-bit exponent addition.

That multiplier design used an array type functional layout similar to the multiplier designs of this thesis. A 16-bit unsigned integer multiplier array required approximately 99,328 $mils^2$ (256 mils by 388 mils) if fabricated with a 1.5 micron feature size. This area requirement is equal to 64.1 mm^2. The Genesil layout operating speed was estimated at 38 MHz. In comparison, the 16-bit fixed point multiplier design example developed in this thesis (scaled to a 1.5 micron feature size) requires 3.08 mm^2 and has a predicted maximum clock frequency of 48.8 MHz with a symmetric clock waveform.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY AND CONCLUSIONS

The purpose of this thesis was to determine the appropriateness and feasibility of using full-custom design methods to produce pipelined adders, subtracters, and multipliers for FFT type operations. The full-custom scaleable CMOS cell system developed produces high speed design layouts compact enough to put an entire radix-2 FFT butterfly on a single VLSI chip. The Genesil Silicon Compiler system, using its present library, was unable to produce designs compact enough to meet that objective.

The cell system developed in this thesis allows a system designer to evaluate the layout size requirements and difficulty tradeoffs for various mathematical operations in both fixed and floating number systems. The system enables a designer to make educated choices as to feature size and system characteristics before starting a layout process. The system will reduce the workload required to complete a layout, but it will by no means eliminate it. The process of layout verification, simulation, and testing will also be very time consuming.

B. RECOMMENDATIONS

Due to the difficulty of generating random logic control functions, design verification, timing simulations, and clock and bus routings on a limited production full-custom design, the cell system developed in this thesis may not be appropriate for direct implementation. Silicon compilers were developed to overcome the shortcomings of the more time consuming layout methods. Several

silicon compiler systems have the capability to allow a user to custom design library cells. Genesil's Genport system, not currently licensed at the Naval Postgraduate School, provides IC design experts with the capability to extend the Genesil Compiler Library with fully parameterized cells that work with Genesil verification and floorplanning tools [Ref. 24]. The capability of the Genport system should allow incorporation of the basic cell designs developed in this thesis into the Genesil library. This would greatly increase the ability to verify designs and aid in wiring and pad placement of the chip.

Designs ported into Genesil loose their flavorlessness and scalability; a specific fabrication line technology and feature size must be chosen as part of the import process. This is not a significant disadvantage; a designer can predict the size of the layouts and the pipeline throughput by examining the characteristics of the cell system developed in this thesis. A choice of feature size and fabrication line can then be made before the cells are ported into Genesil. As an added benefit, Genesil is supported by fabrication lines that can produce radiation hardened devices; MOSIS currently has no radiation-hardened circuit fabrication capability.

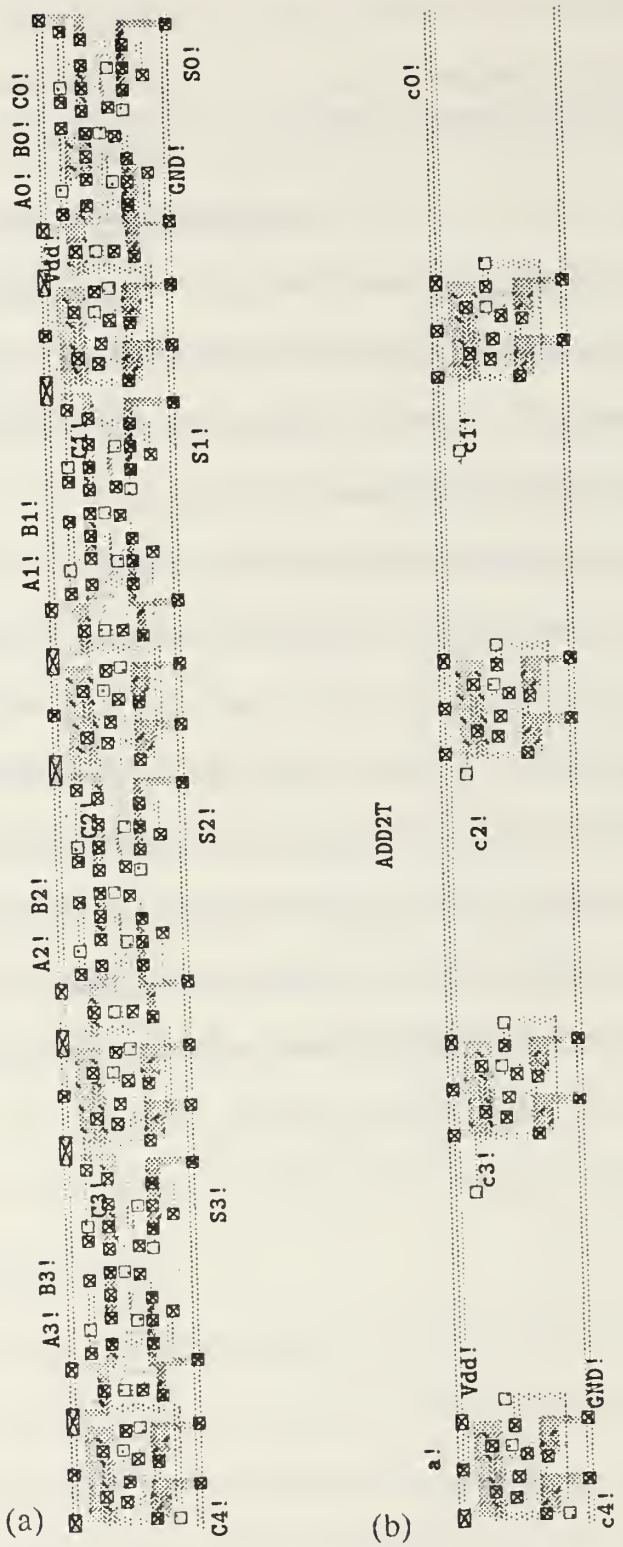


Figure 39. Four adder cell layouts.

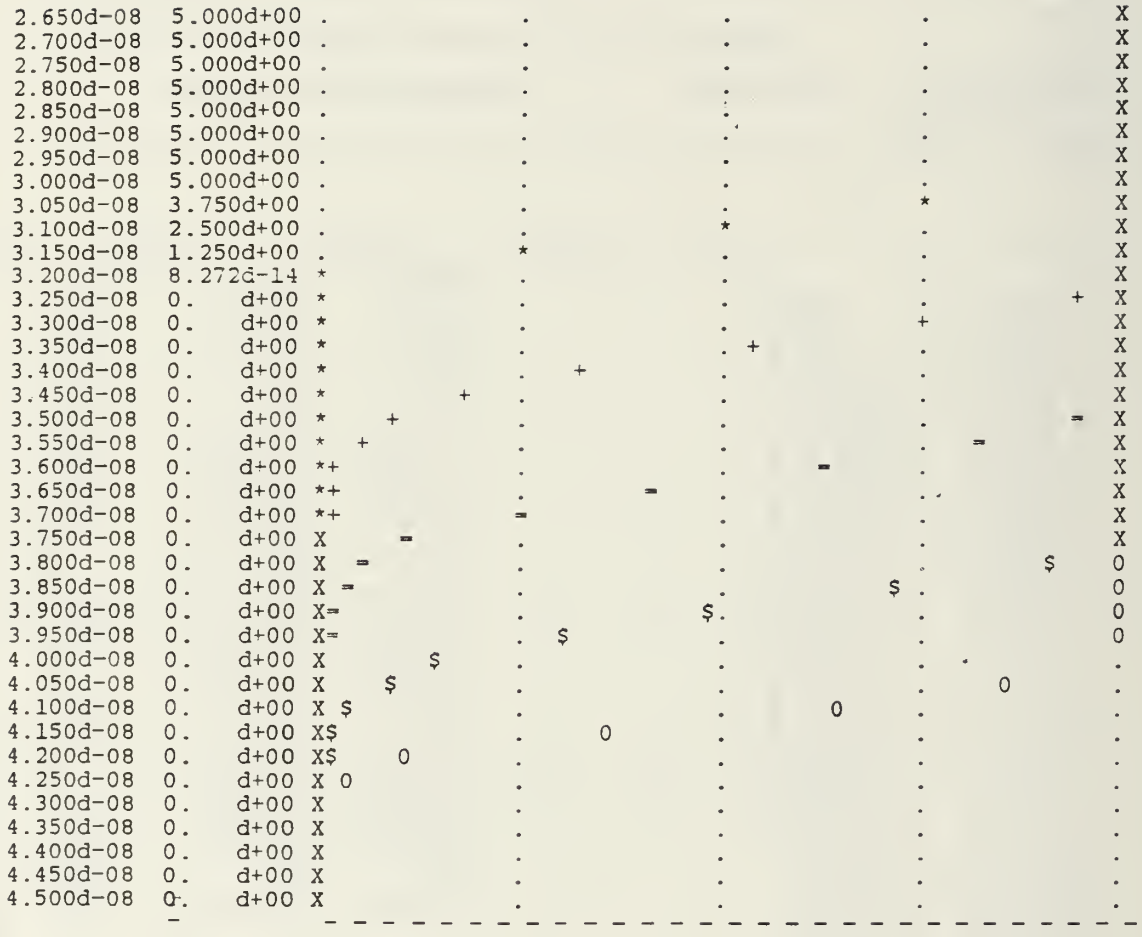
APPENDIX A. SPICE SIMULATION EXAMPLE

All Spice simulations were performed with *lambda* = 1.5 *microns* (3.0 *micron* feature size) and VDD equal to +5.0 volts. Due to Spice's inability to simulate large numbers of transistors, modifications to layouts were made when necessary to obtain results. The example shown is the simulation used to determine the full adder CARRY-OUT delay when only CARRY-IN changes (see Table 4 on page 38). To perform this simulation, four full adder cells were laid out in "series" as shown in Figure 39 (a). After unsuccessful attempts to simulate the circuit, the layout was modified by removing the EXCLUSIVE-OR gates and other logic elements not in the CARRY-IN to CARRY-OUT logic path. The capacitance of this modified layout shown in Figure 39 (b) was adjusted using data from the original circuit to keep the load on logic devices the same as in the unmodified circuit. The Spice deck used for the simulation and the results are shown on the following pages. Note that the results show that the CARRY "ripples" through the circuit with a delay of slightly less than 3 ns per adder cell.

```
*******06/05/90 ********  SPICE 2G.6    3/15/83 ********14:30:07*****
*** SPICE DECK CREATED FROM ADD2T.SIM, TECH=SCMOS
****     MOSFET MODEL PARAMETERS          TEMPERATURE =   27.000 DEG C
******************************************************************************
```

	CMOSP	CMOSN
TYPE	PMOS	NMOS
LEVEL	2.000	2.000
VTO	-0.900	0.900
KP	1.38d-05	3.11d-05
GAMMA	0.590	1.319
PHI	0.660	0.743
CGSO	4.00d-10	5.20d-10
CGDO	4.00d-10	5.20d-10
CGBO	4.00d-10	5.20d-10
RSH	95.000	20.000
CJ	2.00d-04	3.20d-04
MJ	0.500	0.500
CJSW	4.50d-10	9.00d-10
MJSW	0.330	0.330
JS	1.00d-04	1.00d-04
TOX	5.00d-08	5.00d-08
NSUB	5.00d+15	2.50d+16
TPG	-1.000	1.000
XJ	6.00d-07	8.00d-07
LD	5.00d-07	6.40d-07
UO	200.000	450.000
UCRIT	8.00d+04	8.00d+04
UEXP	0.150	0.150
UTRA	0.300	0.300
VMAX	5.00d+04	5.00d+04

```
*******06/05/90 ********  SPICE 2G.6    3/15/83 ********14:30:07*****

*** SPICE DECK CREATED FROM ADD2T.SIM, TECH=SCMOS

****      TRANSIENT ANALYSIS                TEMPERATURE =   27.000 DEG C

******************************************************************************


LEGEND:

 *: V(17)= C0
 +: V(15)= C1
 =: V(10)= C2
 $: V(8) = C3
 0: V(6) = C4
X
     TIME       V(17)

(*+=$0)---------  0.    d+00      1.250d+00      2.500d+00      3.750d+00  5.000d+00
                  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 9.000d-09  0.    d+00 X              .              .              .              .
 9.500d-09  0.    d+00 X              .              .              .              .
 1.000d-08  0.    d+00 X              .              .              .              .
 1.050d-08  1.250d+00  X              *              .              .              .
 1.100d-08  2.500d+00  X              .              *              .              .
 1.150d-08  3.750d+00  X              .              .              *              .
 1.200d-08  5.000d+00  X              .              .              .              *
 1.250d-08  5.000d+00  X  +           .              .              .              *
 1.300d-08  5.000d+00  X          +   .              .              .              *
 1.350d-08  5.000d+00  X              .       +      .              .              *
 1.400d-08  5.000d+00  X              .              .      +       .              *
 1.450d-08  5.000d+00  X              .              .              .+             *
 1.500d-08  5.000d+00  X=             .              .              .         +    *
 1.550d-08  5.000d+00  X      -       .              .              .             + *
 1.600d-08  5.000d+00  X              =              .              .             + *
 1.650d-08  5.000d+00  X              .              =              .             +*
 1.700d-08  5.000d+00  X              .              .      -       .             +*
 1.750d-08  5.000d+00  X              .              .              .  -          X
 1.800d-08  5.000d+00  0 $            .              .              .         =    X
 1.850d-08  5.000d+00  0       $      .              .              .           =  X
 1.900d-08  5.000d+00  0              .   $          .              .             - X
 1.950d-08  5.000d+00  0              .              $              .             =X
 2.000d-08  5.000d+00  0              .              .      $       .             =X
 2.050d-08  5.000d+00  0              .              .              .   $         X
 2.100d-08  5.000d+00  .    0         .              .              .         $    X
 2.150d-08  5.000d+00  .              0.             .              .           $ X
 2.200d-08  5.000d+00  .              .              .0             .             $X
 2.250d-08  5.000d+00  .              .              .              0             $X
 2.300d-08  5.000d+00  .              .              .              .         0   $X
 2.350d-08  5.000d+00  .              .              .              .             0X
 2.400d-08  5.000d+00  .              .              .              .             X
 2.450d-08  5.000d+00  .              .              .              .             X
 2.500d-08  5.000d+00  .              .              .              .             X
 2.550d-08  5.000d+00  .              .              .              .             X
 2.600d-08  5.000d+00  .              .              .              .             X
```


2.650d-08 5.000d+00
2.700d-08 5.000d+00
2.750d-08 5.000d+00
2.800d-08 5.000d+00
2.850d-08 5.000d+00
2.900d-08 5.000d+00
2.950d-08 5.000d+00
3.000d-08 5.000d+00
3.050d-08 3.750d+00
3.100d-08 2.500d+00
3.150d-08 1.250d+00
3.200d-08 8.272d-14
3.250d-08 0. d+00
3.300d-08 0. d+00
3.350d-08 0. d+00
3.400d-08 0. d+00
3.450d-08 0. d+00
3.500d-08 0. d+00
3.550d-08 0. d+00
3.600d-08 0. d+00
3.650d-08 0. d+00
3.700d-08 0. d+00
3.750d-08 0. d+00
3.800d-08 0. d+00
3.850d-08 0. d+00
3.900d-08 0. d+00
3.950d-08 0. d+00
4.000d-08 0. d+00
4.050d-08 0. d+00
4.100d-08 0. d+00
4.150d-08 0. d+00
4.200d-08 0. d+00
4.250d-08 0. d+00
4.300d-08 0. d+00
4.350d-08 0. d+00
4.400d-08 0. d+00
4.450d-08 0. d+00
4.500d-08 0. d+00

APPENDIX B. STANDARD CELL DESCRIPTIONS AND LAYOUTS

A. EXPONENT ADDITION FUNCTION CELLS

The standard cells used to perform the exponent addition function used in a floating point multiplication are listed below. These cells not only sum the two exponents, but also simultaneously produce a modified sum by subtracting 1 from the sum. These two values are then stored in latches until the result of the multiply is completed and the appropriate value can be "selected". The arrangement of the cells is shown in Figure 38 on page 66. The following list describes the cells and their functions:

Cell AA: Figure 40 on page 80; a standard full adder cell. The inputs are labeled A, B, and Ci (CARRY-IN); the outputs are Co (CARRY-OUT) and S (SUM). Note that a clock line (CLK) passes through the cell although it is unused by the cell. Also note that a ground bus (GND) runs along the lower edge of the cell. The cell's connections to the positive bus (VDD) are "made" by placing the cell directly below a cell with the VDD bus running along its lower edge.

Cell AAL: Figure 41 on page 81; a full adder cell with the same inputs and outputs as cell AA. The cell has a slightly modified layout and is designed to be used as the last logic cell before a set of pipeline latches.

Cell AAR: Figure 42 on page 82; a full adder cell produced by modifying cell AA. It is designed to be the first full adder cell used in an addition process. The CARRY-IN logic line is tied to ground (a logic zero).

Cell AA1: Figure 43 on page 83; a full adder cell designed to be the first adder cell in a pipeline stage. The CARRY-IN line is arranged to provide a connection to a latch instead of a previous adder cell.

Cell AB: Figure 44 on page 84; a standard full adder cell. The VDD bus runs along the lower edge of the cell. The cell must be placed directly below a cell with a GND bus running along its lower edge.

Cell AB1: Figure 45 on page 85; a full adder cell designed to be the first adder cell in a pipeline stage. The CARRY-IN line is arranged to provide a connection to a latch instead of a previous adder cell.

Cell CA: Figure 46 on page 86; a "clock" cell that provides a CLK output to cells in its row by inverting the CLKin signal. This cell has a GND bus running along its lower edge as well as GND and VDD busses running vertically.

Cell CB: Figure 47 on page 87; a clock cell similar to cell CA, but with a VDD bus running along its lower edge.

Cell CI: Figure 48 on page 88); a clock cell designed to be used at the top of the layout. It has a GND bus running along its top edge and a VDD bus running along its lower edge.

Cell ENA: Figure 49 on page 89; a "two-level" cell that performs the function of subtracting 1 from the exponent sum. Figure 27 on page 51 shows the gate level logic of the subtraction process. The signal S is the sum produced by one of the full adder cells. The signal Sm is the modified sum (sum-1). The CARRY-IN (Ci) and CARRY-OUT (Co) signals are used in the subtraction process. Cell ENA also contains four latches which store S and Sm for two clock cycles.

Cell ENAL: Figure 50 on page 90; similar to cell ENA, but designed to be the last logic cell before a pipeline latch stage. To prevent the logic from being in the critical timing path, this cell latches S and Ci prior to performing the subtraction function.

Cell ENAW: Figure 51 on page 91; similar to cell ENA, but designed to be the first logic cell after a pipeline latch stage.

Cell ENB: Figure 52 on page 92; similar to cell ENA in function, but has the VDD bus running across its center and the GND bus running along its lower edge.

Cell ENBL: Figure 53 on page 93; similar to cell ENB, but designed to be the last logic cell before a pipeline latch stage. To prevent the logic from being in the critical timing path, this cell stores S and Ci in latches prior to performing the subtraction function.

Cell ENBR: Figure 54 on page 94; designed as the first cell in the subtract 1 from exponent sum process. Sm is the inverse of S. Other functions remain the same as in cell ENB.

Cell ENBW: Figure 55 on page 95; similar to cell ENB, but designed to be the first logic cell after a pipeline latch stage.

Cell RA: Figure 56 on page 96; a pipeline latch cell designed to store two values (A and B). This cell is used both to store the exponent bits prior to the addition process, and to store the sum and modified sum (S and Sm) bits after the addition and subtract 1 processes. Cell RA has a GND bus running along its lower edge.

Cell RAI: Figure 57 on page 97; a latch cell designed to be used a the top of the layout. Cell RAI stores two values and has a GND bus running along its top edge and a VDD bus running along its lower edge.

Cell RAIW: Figure 58 on page 98; performs the same function, but is slightly "wider" than cell RAI. The extra width is needed in cells used below RAIW.

Cell RAW: Figure 59 on page 99; a latch cell similar to cell RA, but slightly wider.

Cell RA1: Figure 60 on page 100; a latch cell that stores three values. It is used to the left of cell AAL, and stores the CARRY-OUT for use in the next pipeline logic stage.

Cell RB: Figure 61 on page 101; a latch cell that performs the same function as cell RA, But has a VDD bus running along its lower edge.

Cell RBW: Figure 62 on page 102; a latch cell similar to cell RB, But slightly wider.

Cell RB1: Figure 63 on page 103; a latch cell similar to cell RB, but modified to store three values. (See cell RA1.)

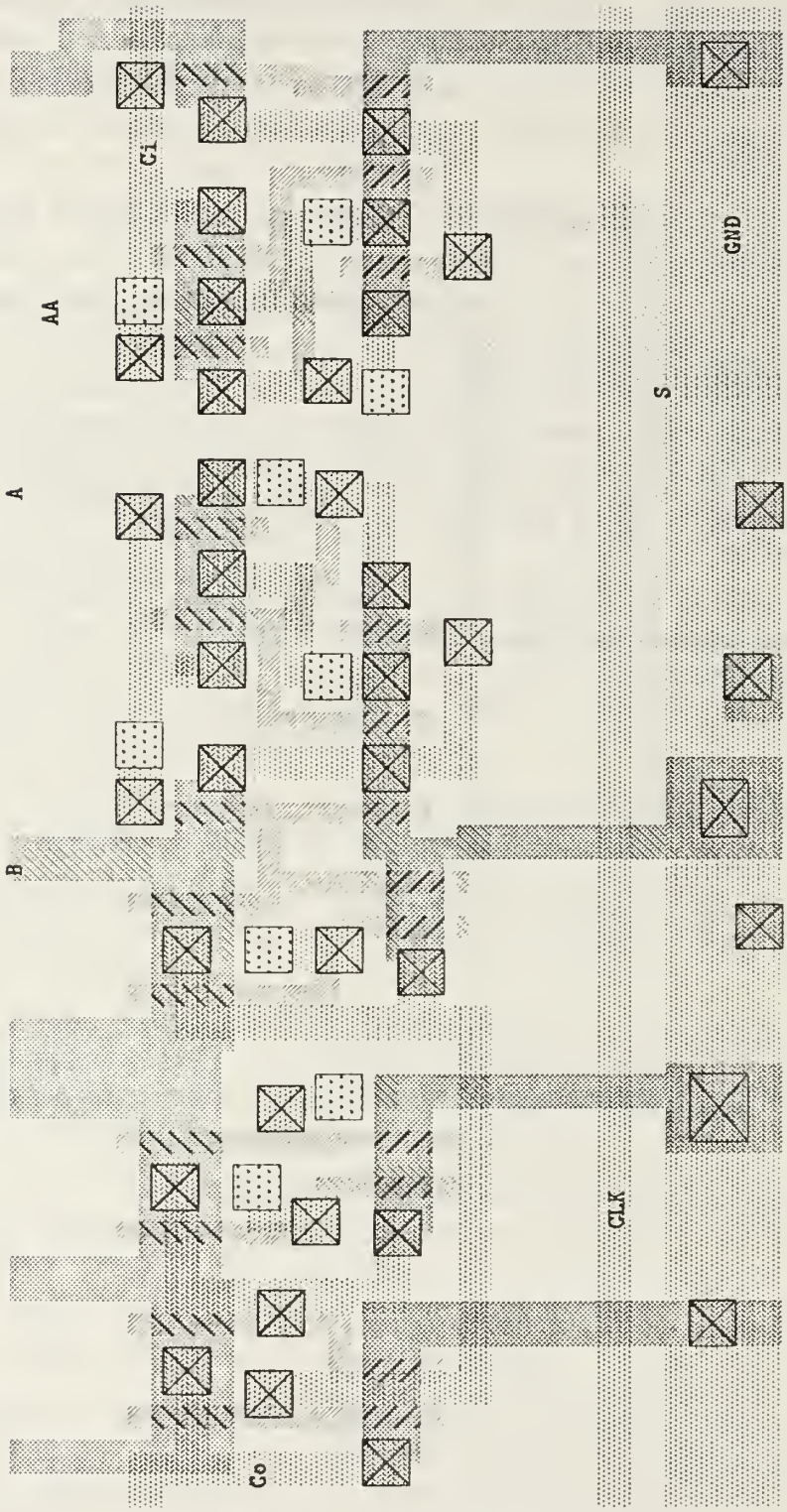


Figure 40. Cell AA layout

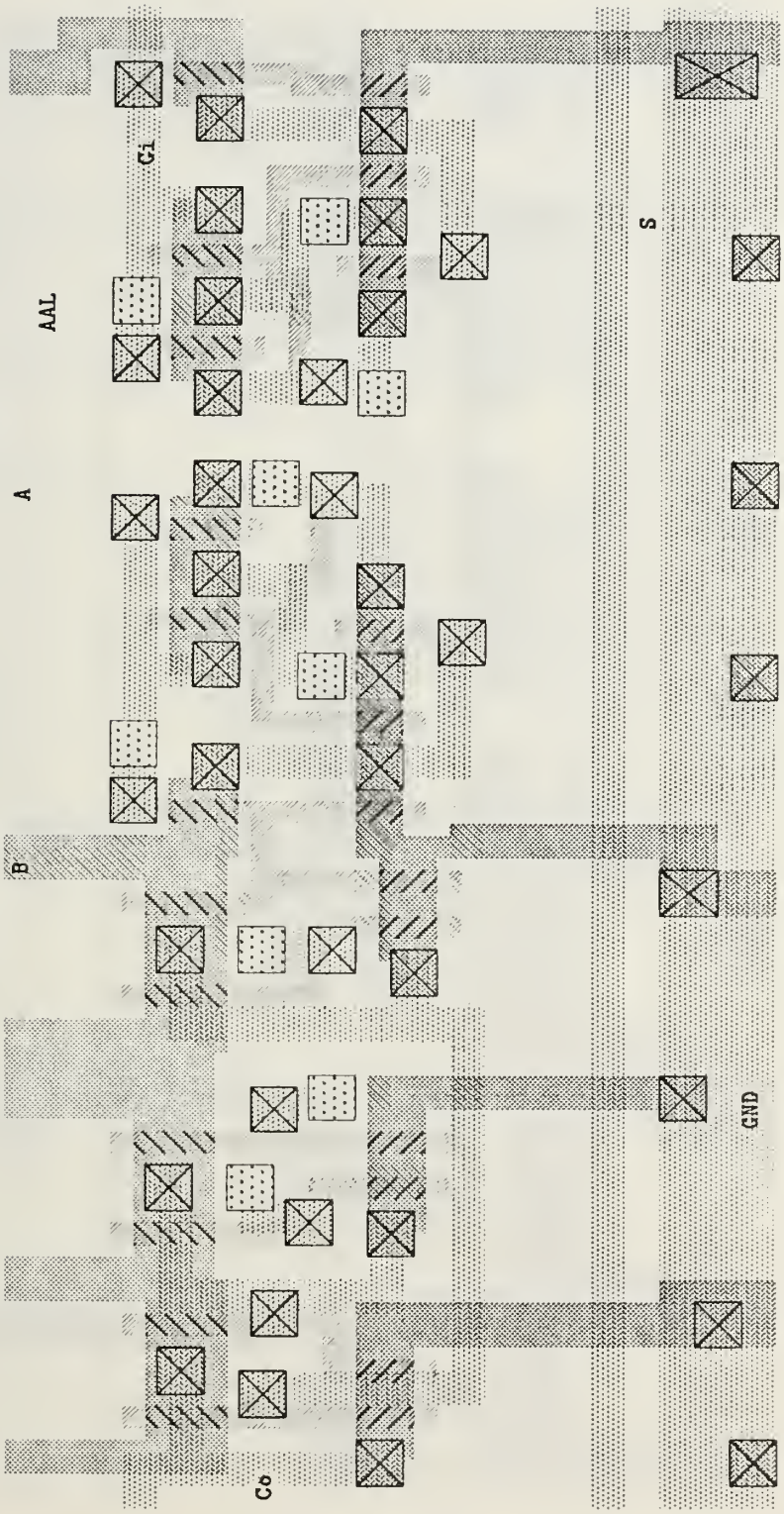


Figure 41. Cell AAL layout

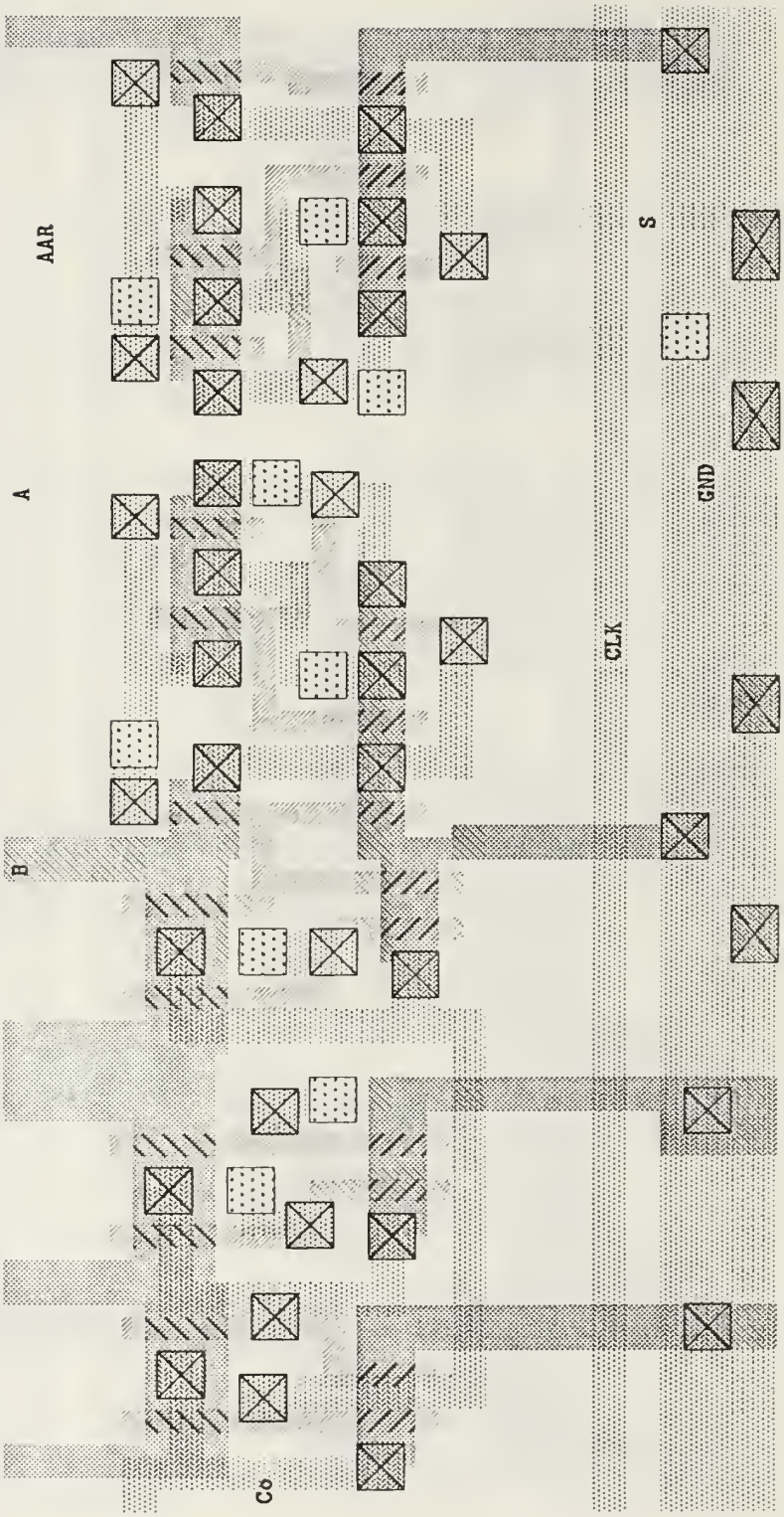


Figure 42. Cell AAR layout

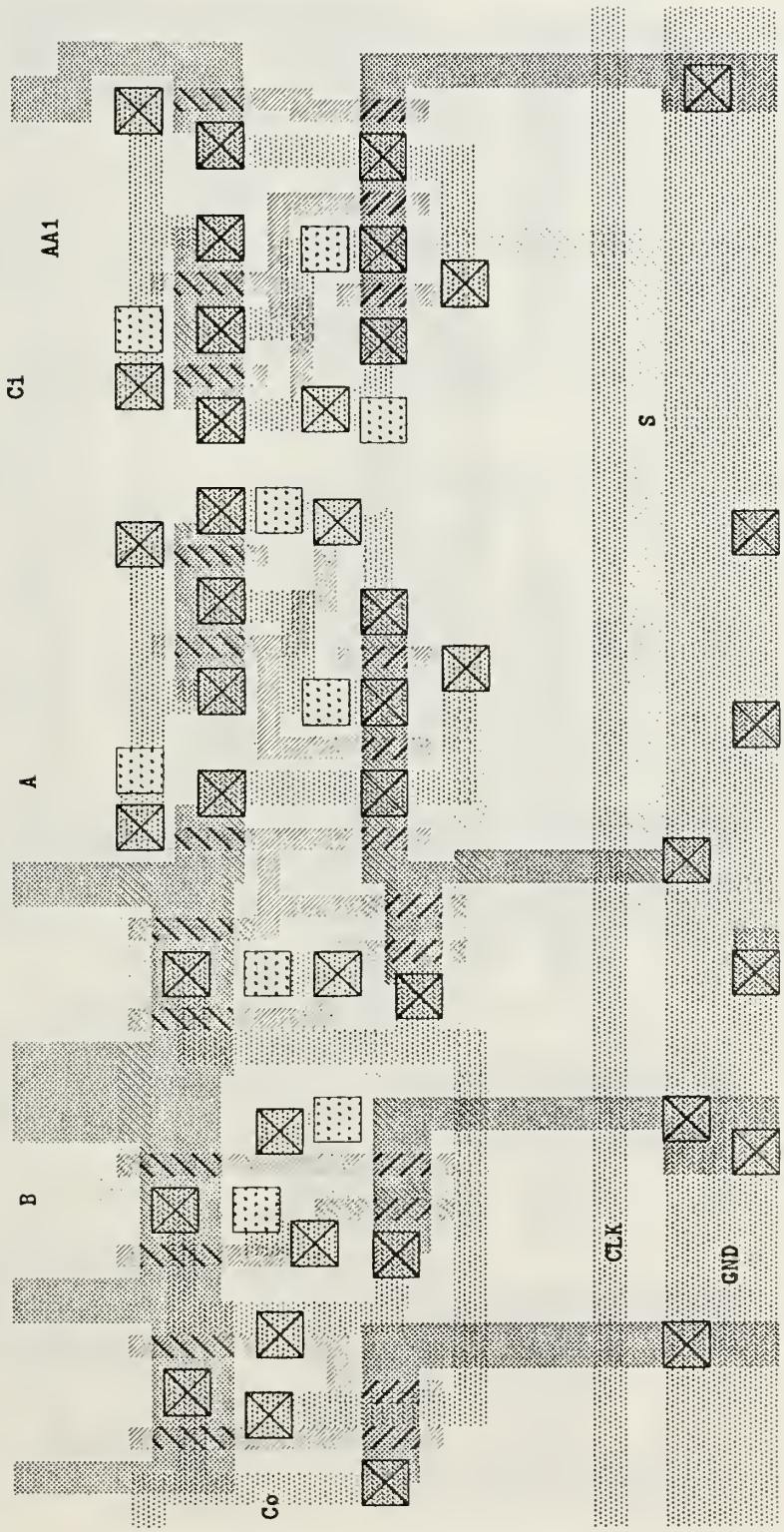


Figure 43. Cell AA1 layout

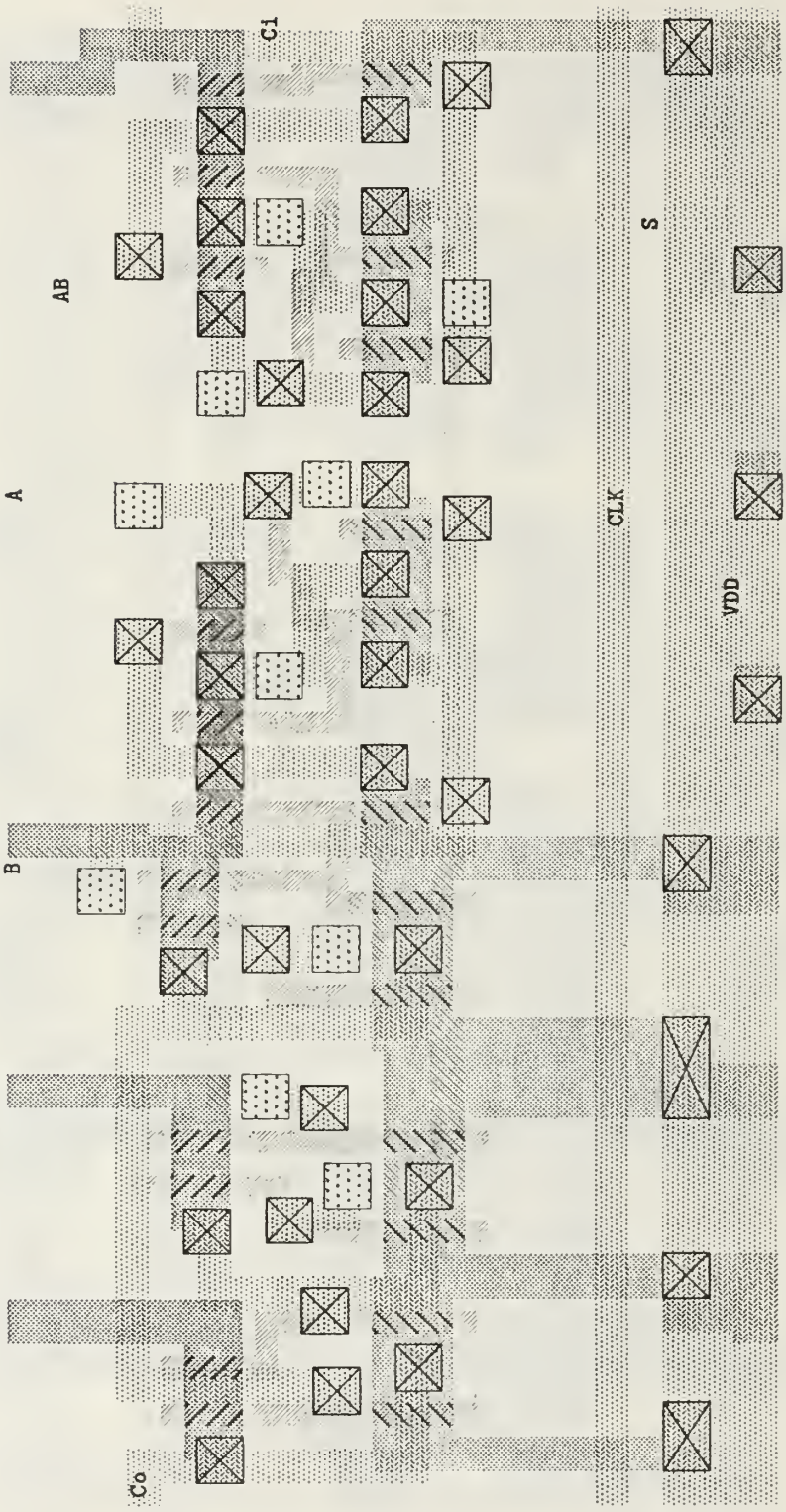


Figure 44. Cell AB layout

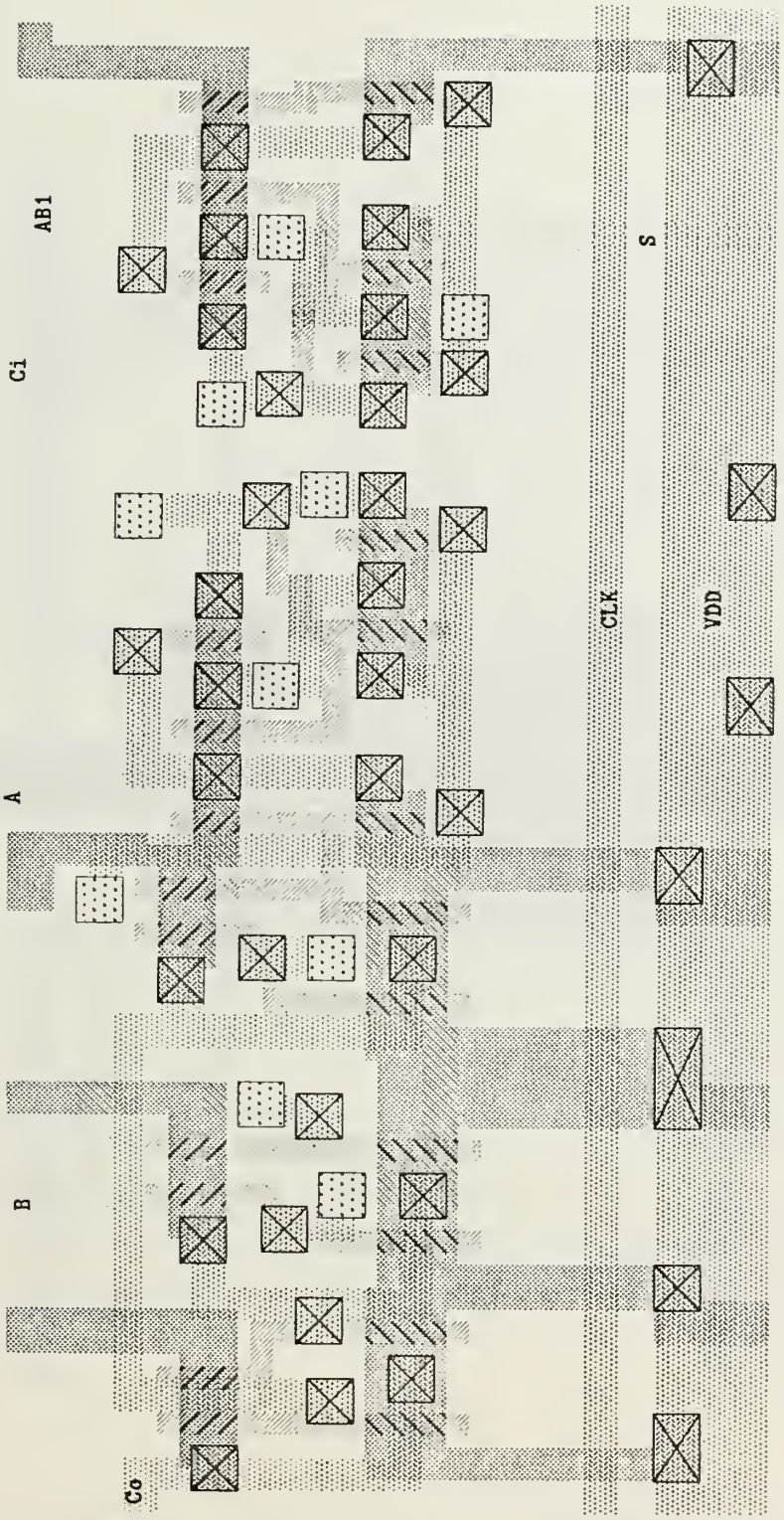


Figure 45. Cell AB1 layout

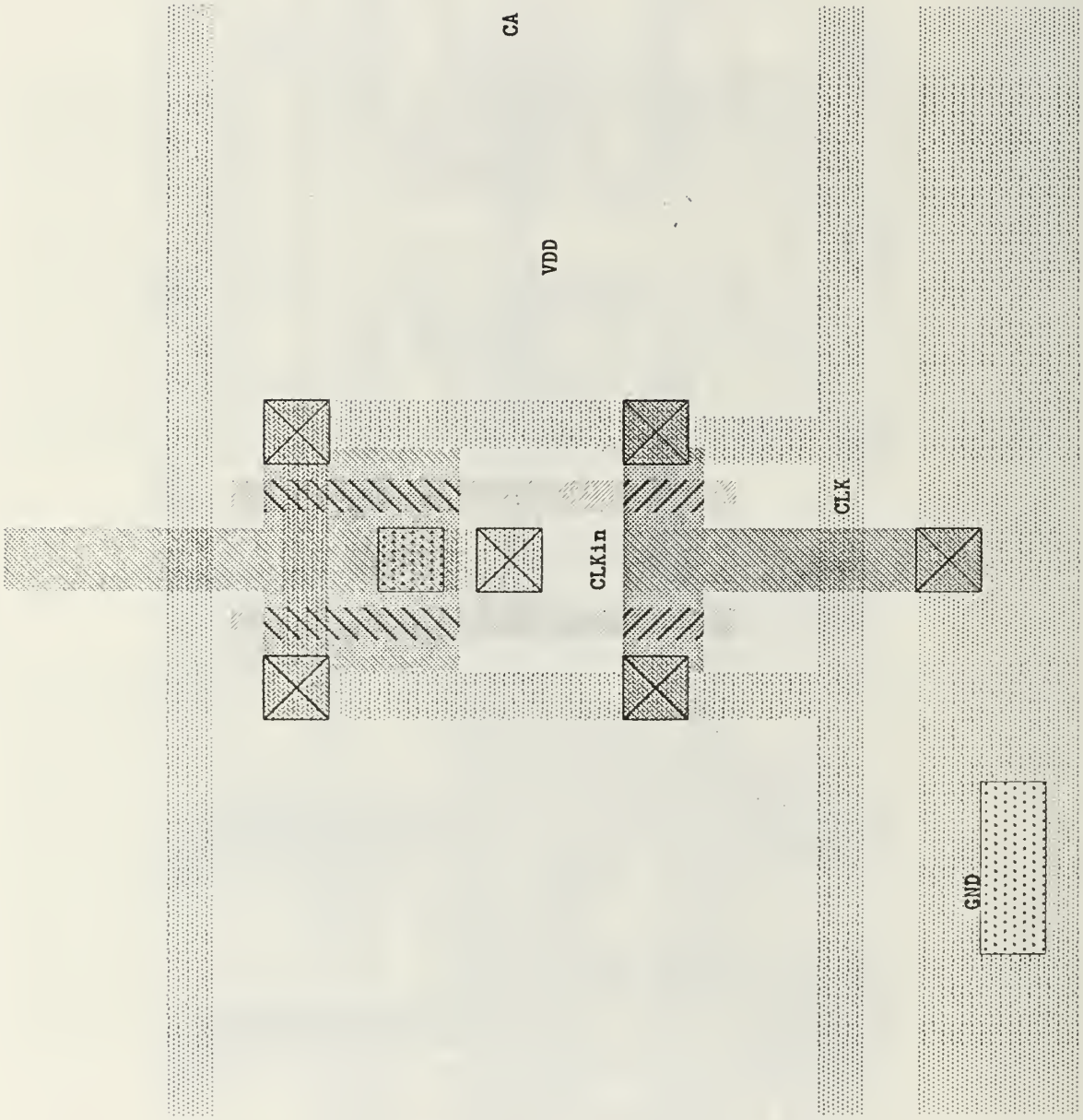


Figure 46. Cell CA layout

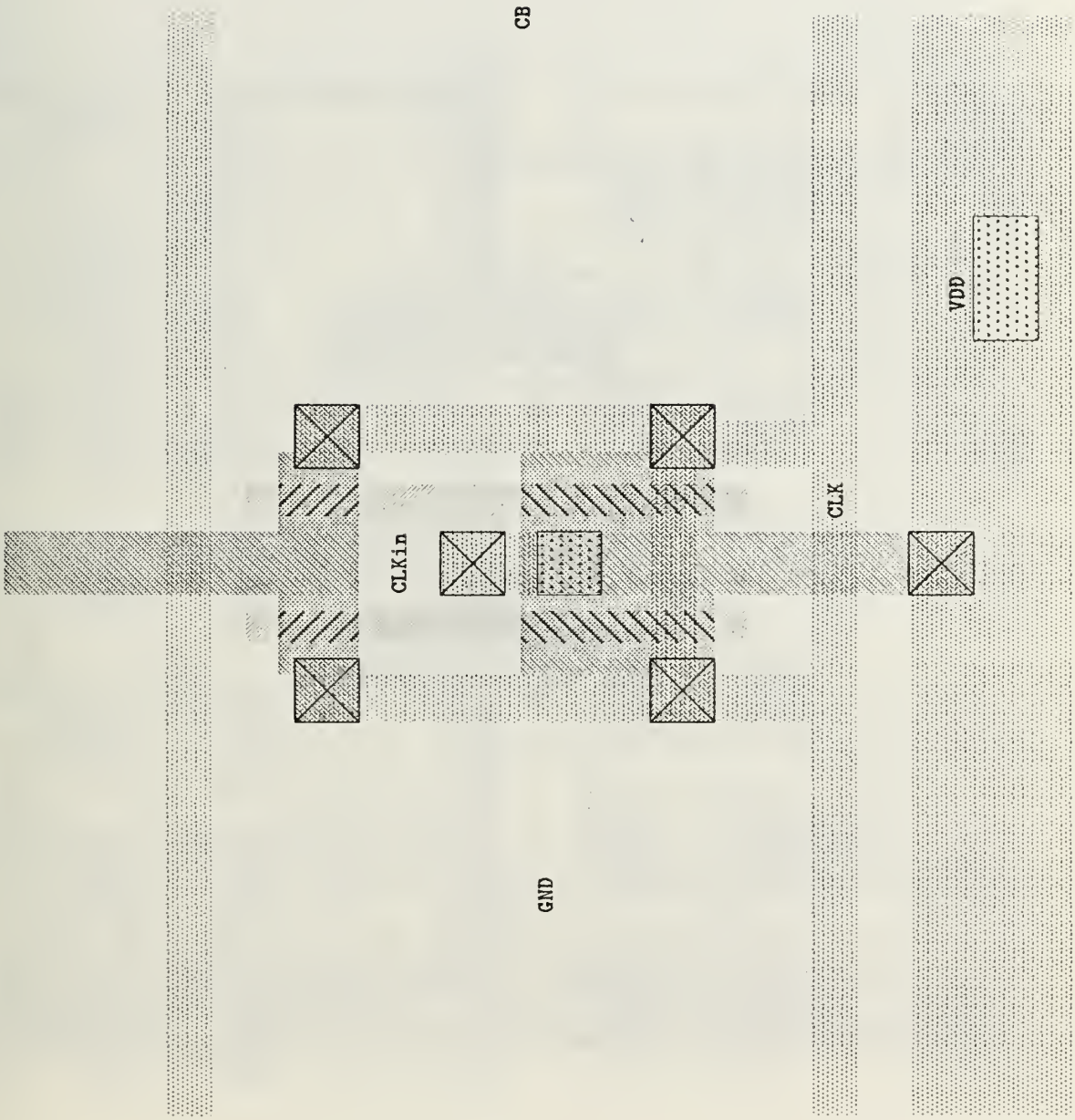


Figure 47. Cell CB layout

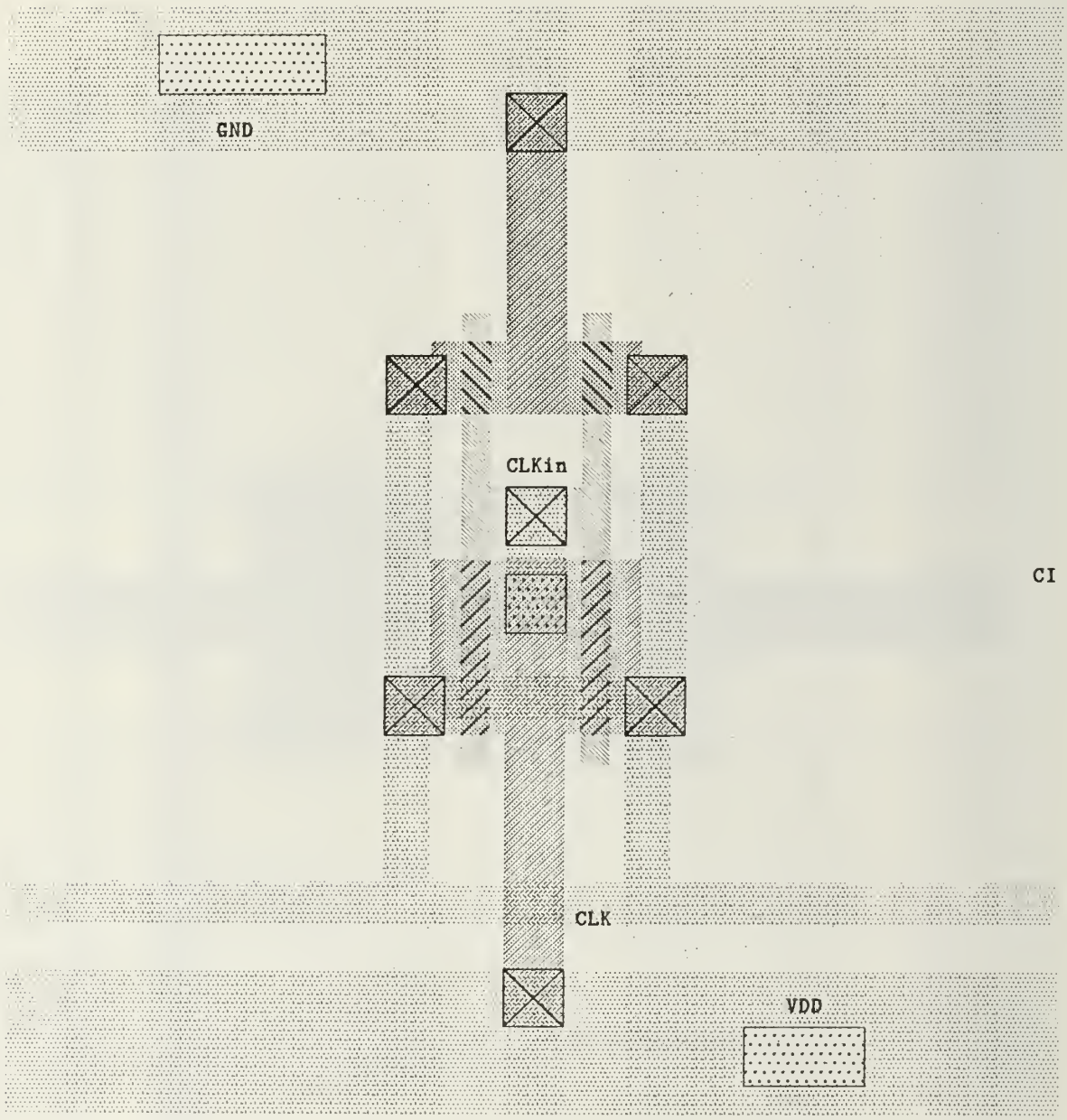


Figure 48. Cell CI layout

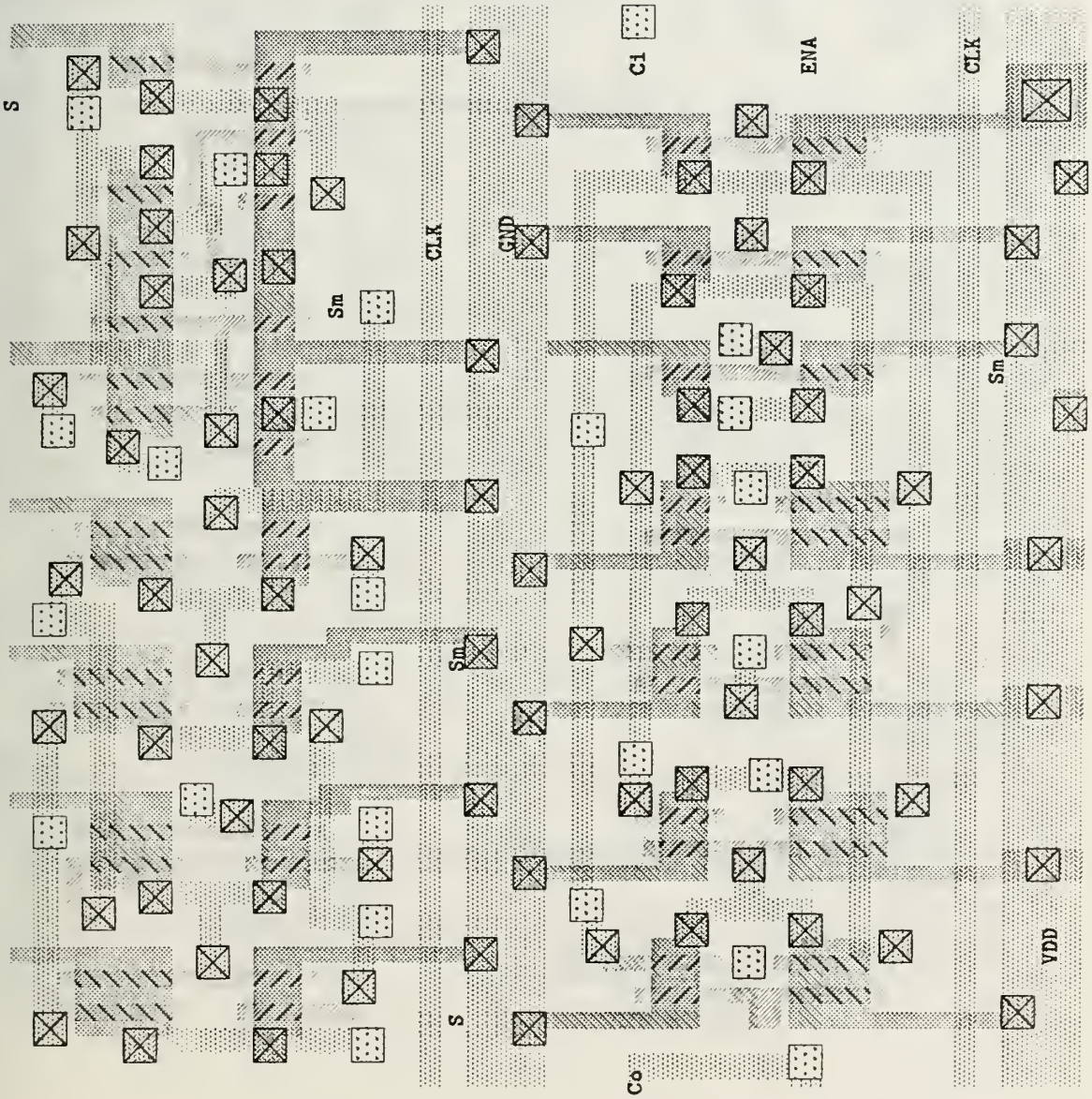


Figure 49. Cell ENA layout

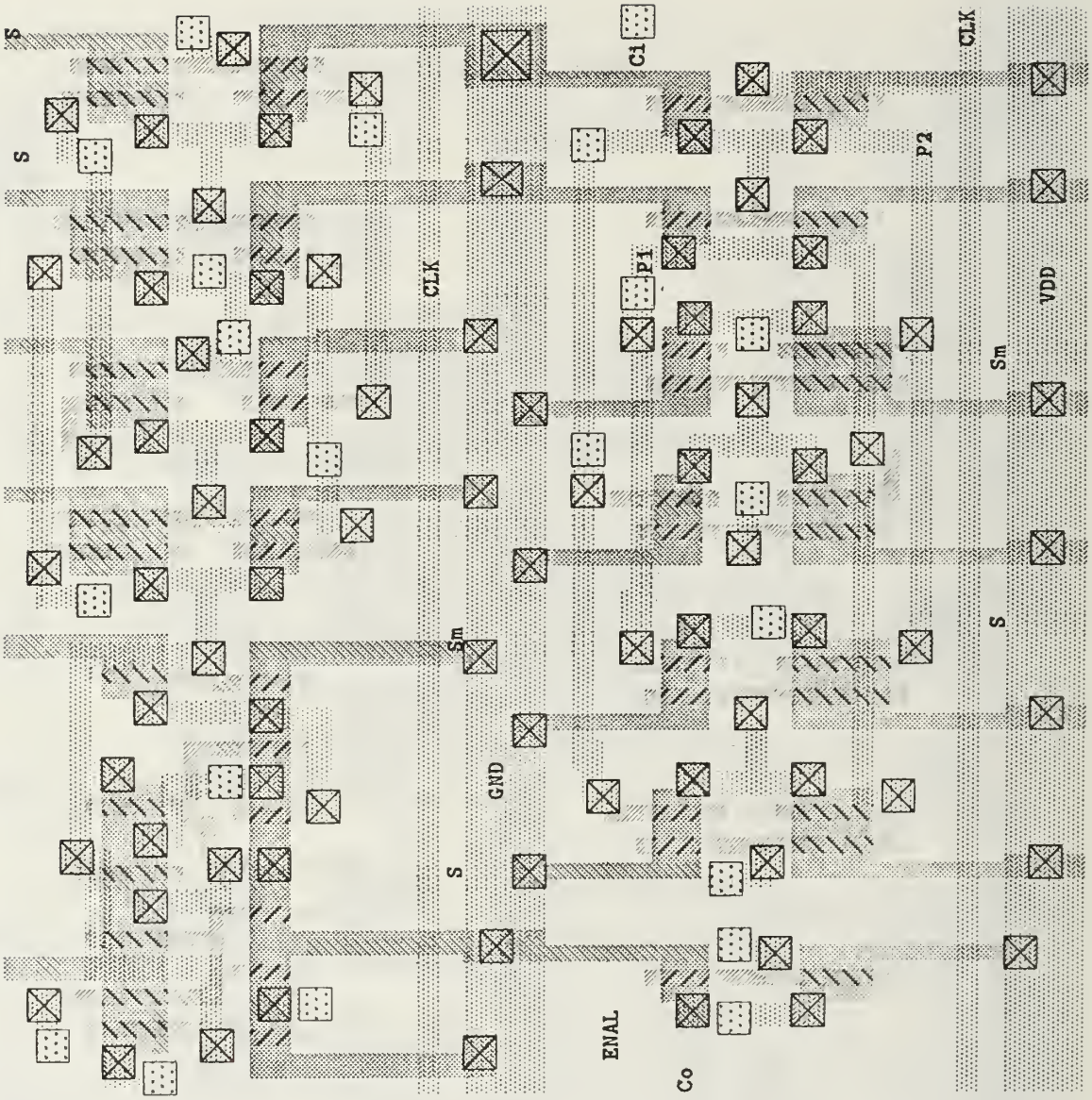


Figure 50. Cell ENAL layout

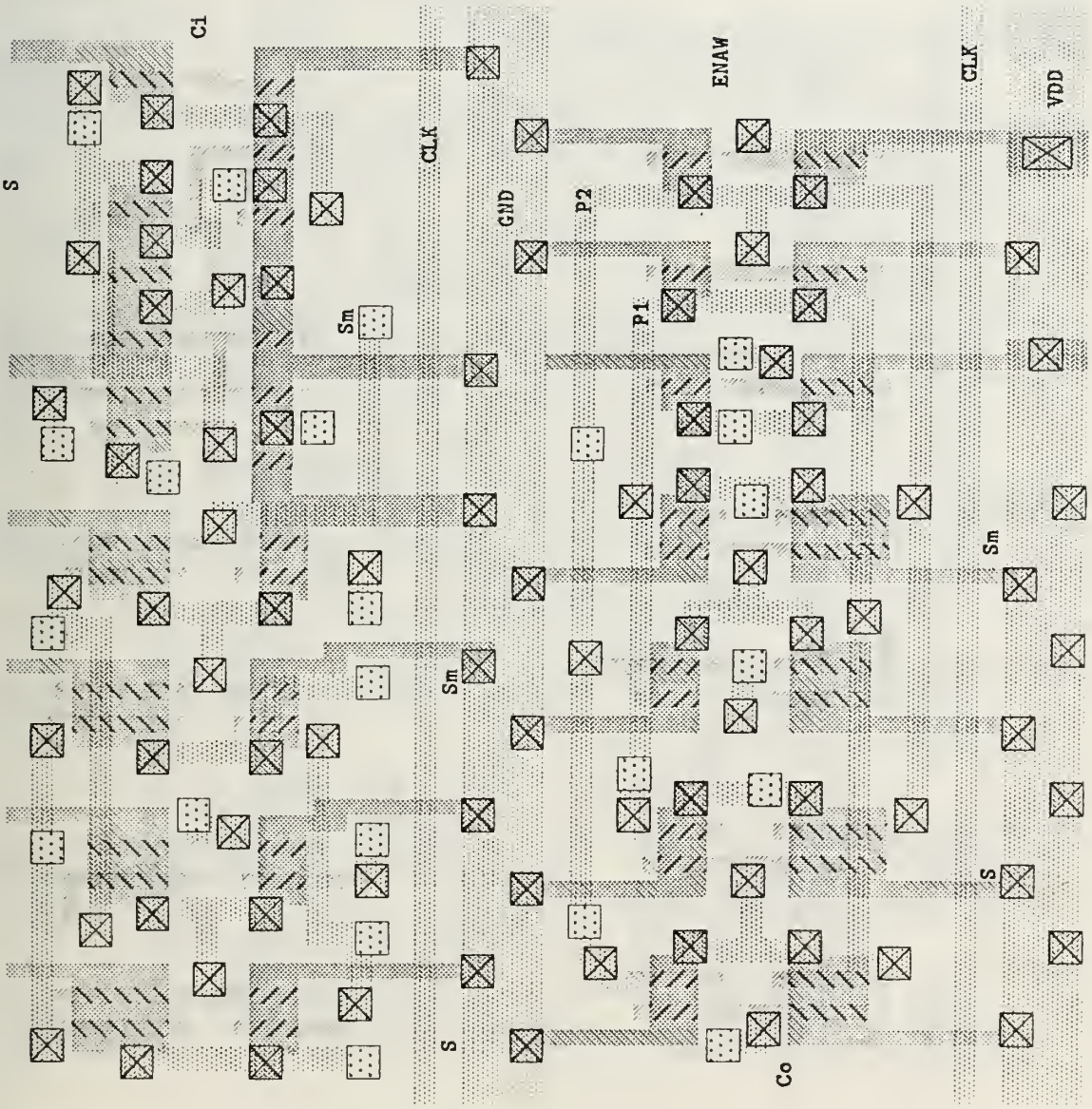


Figure 51. Cell ENAW layout

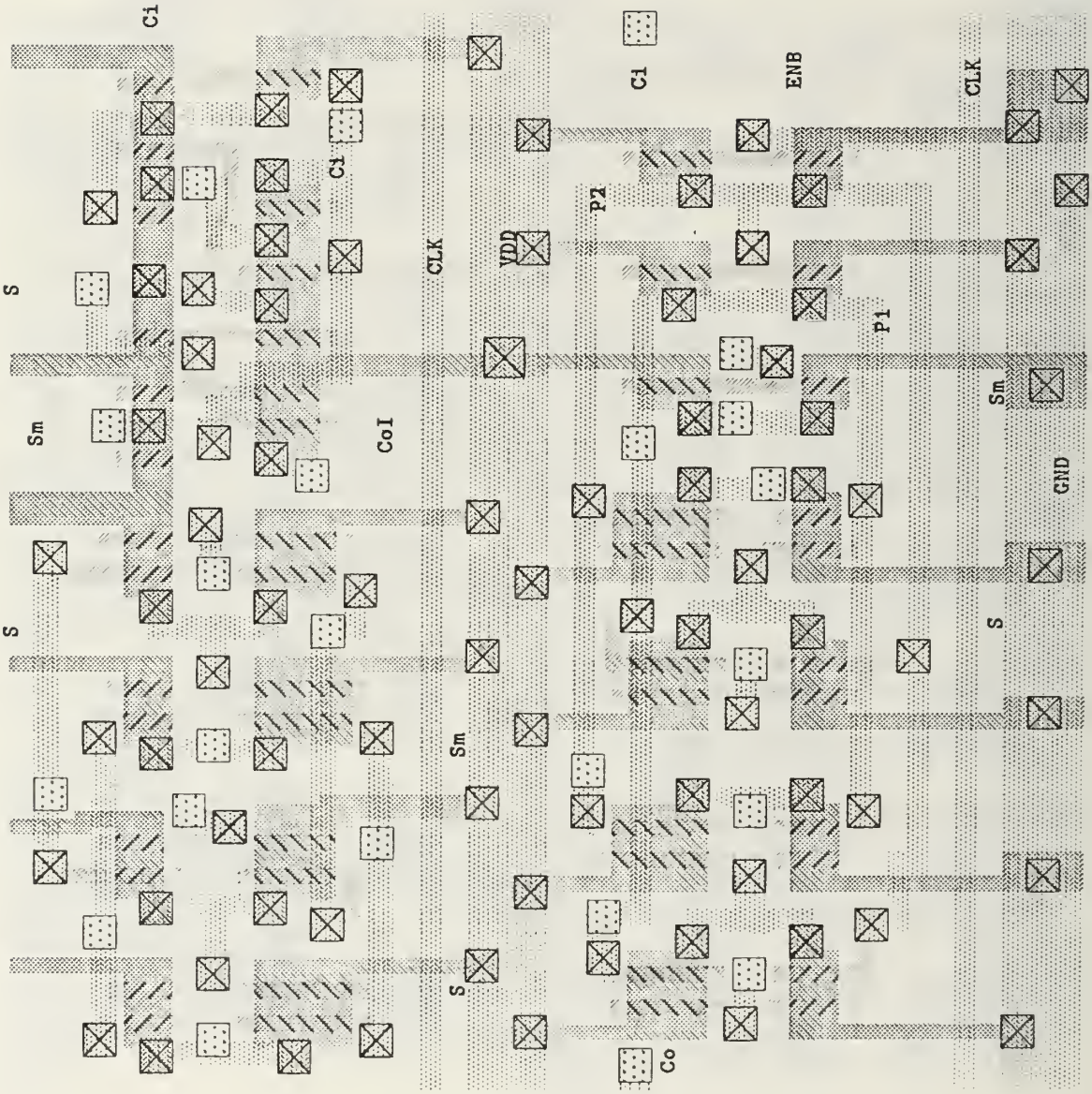


Figure 52. Cell ENB layout

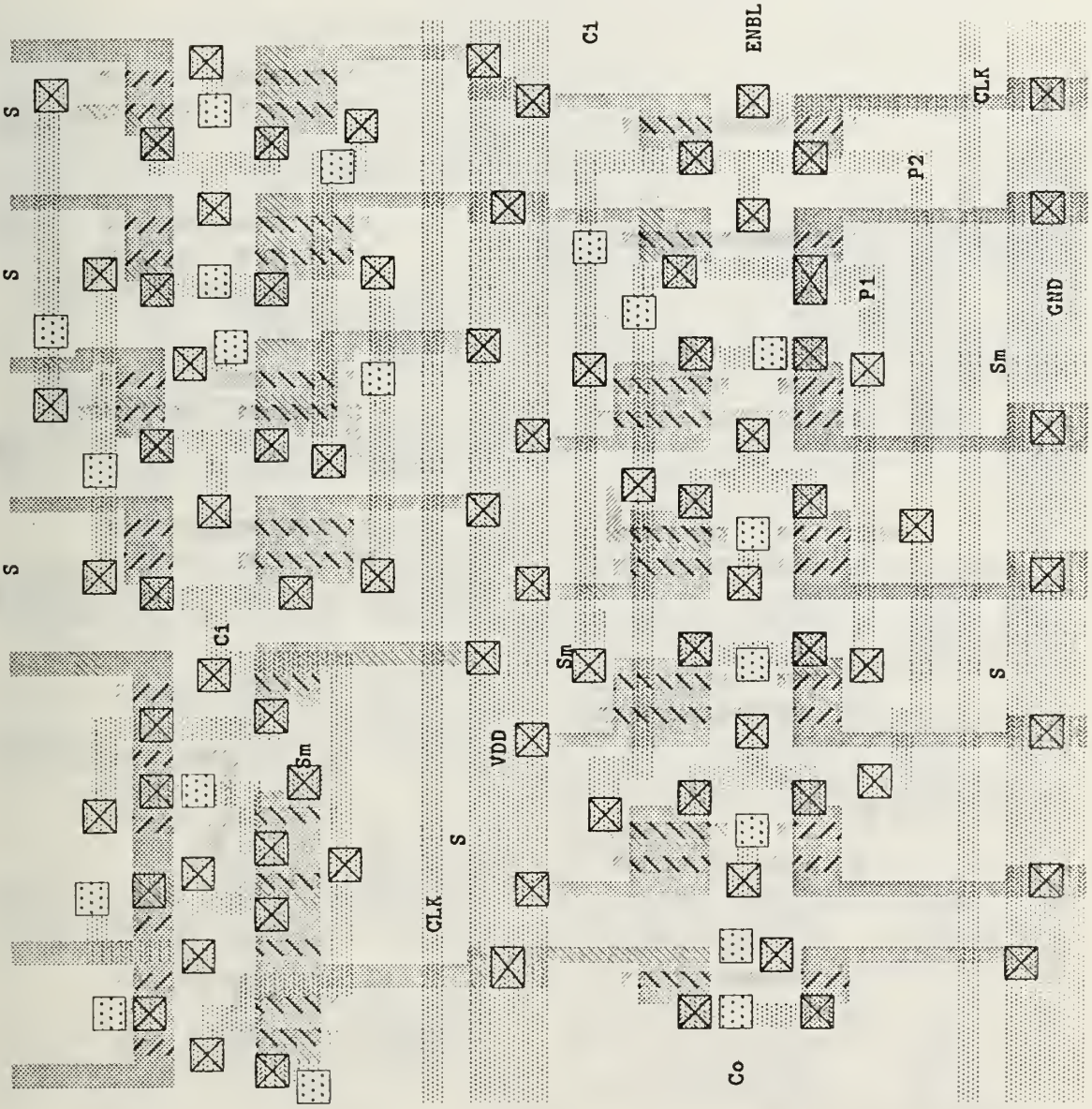


Figure 53. Cell ENBL layout

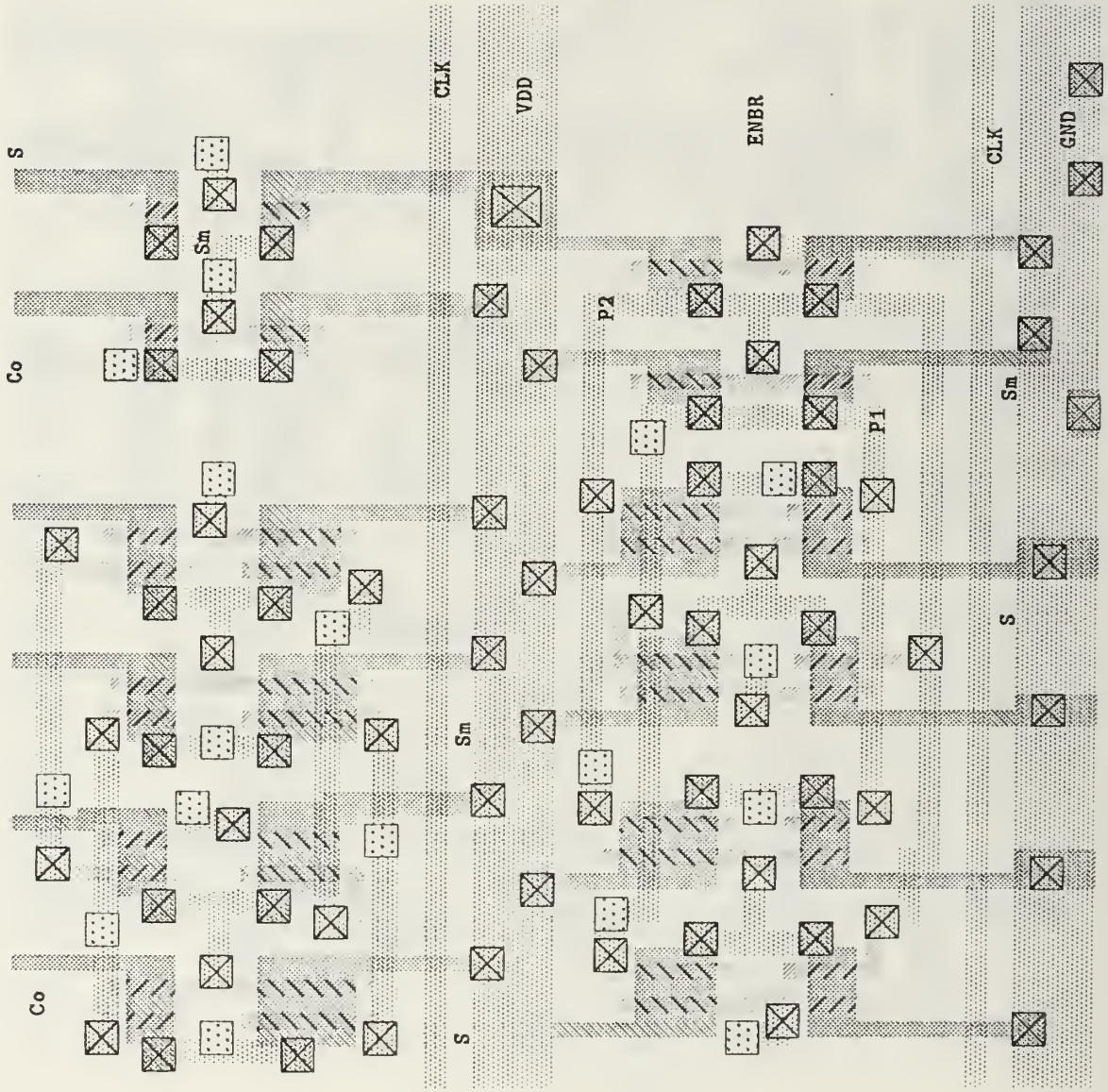


Figure 54. Cell ENBR layout

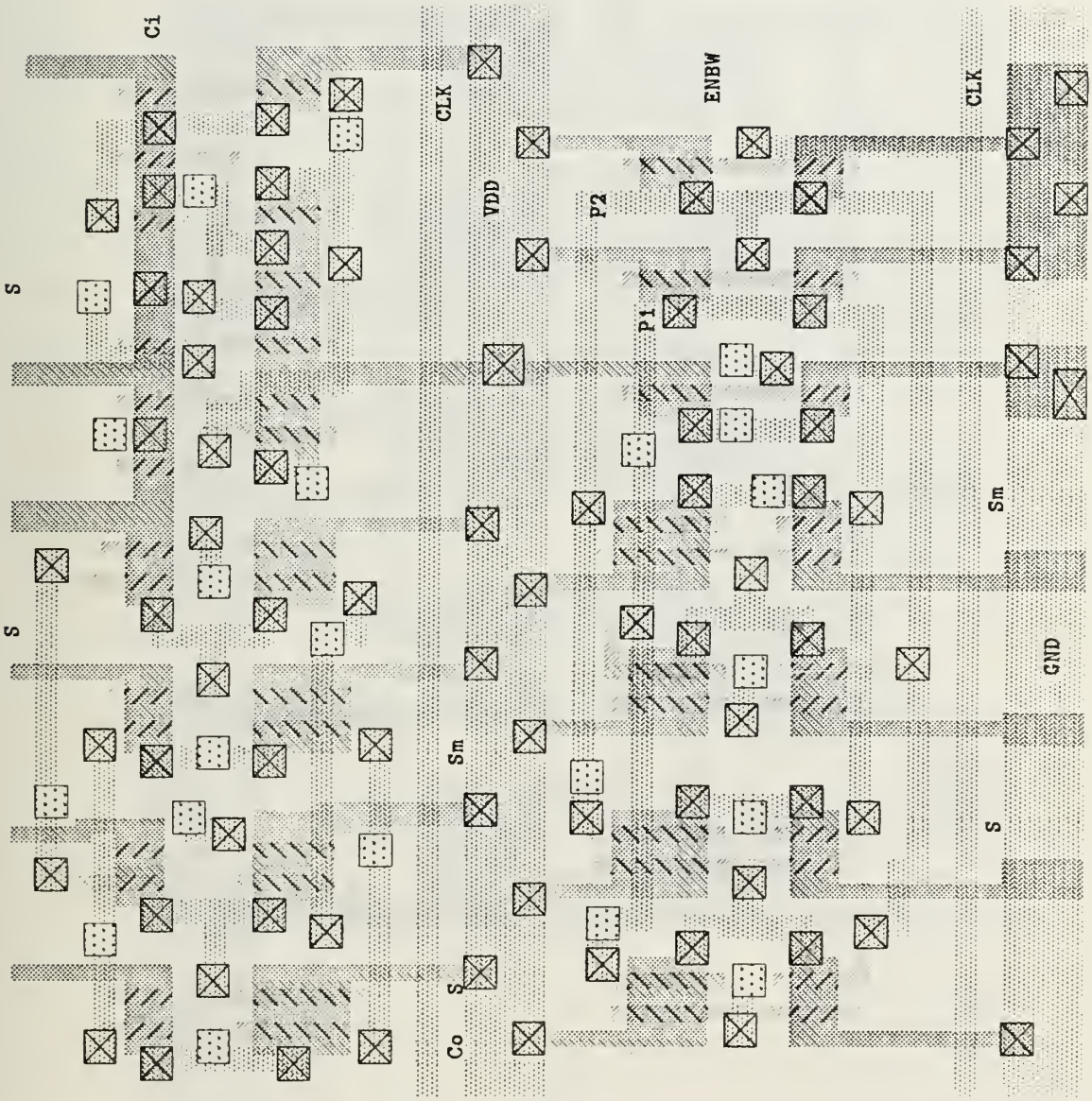


Figure 55. Cell ENBW layout

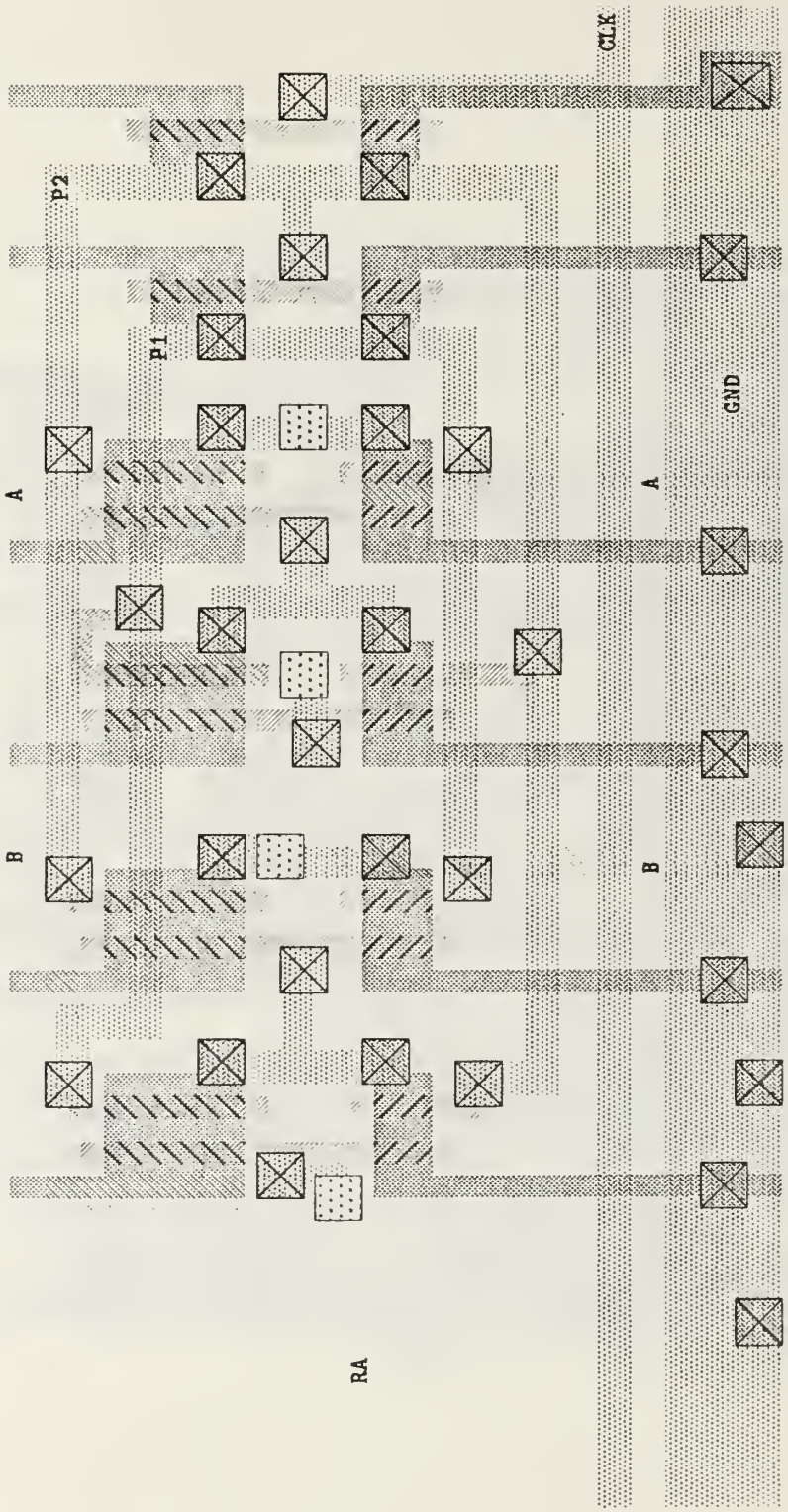


Figure 56. Cell RA layout

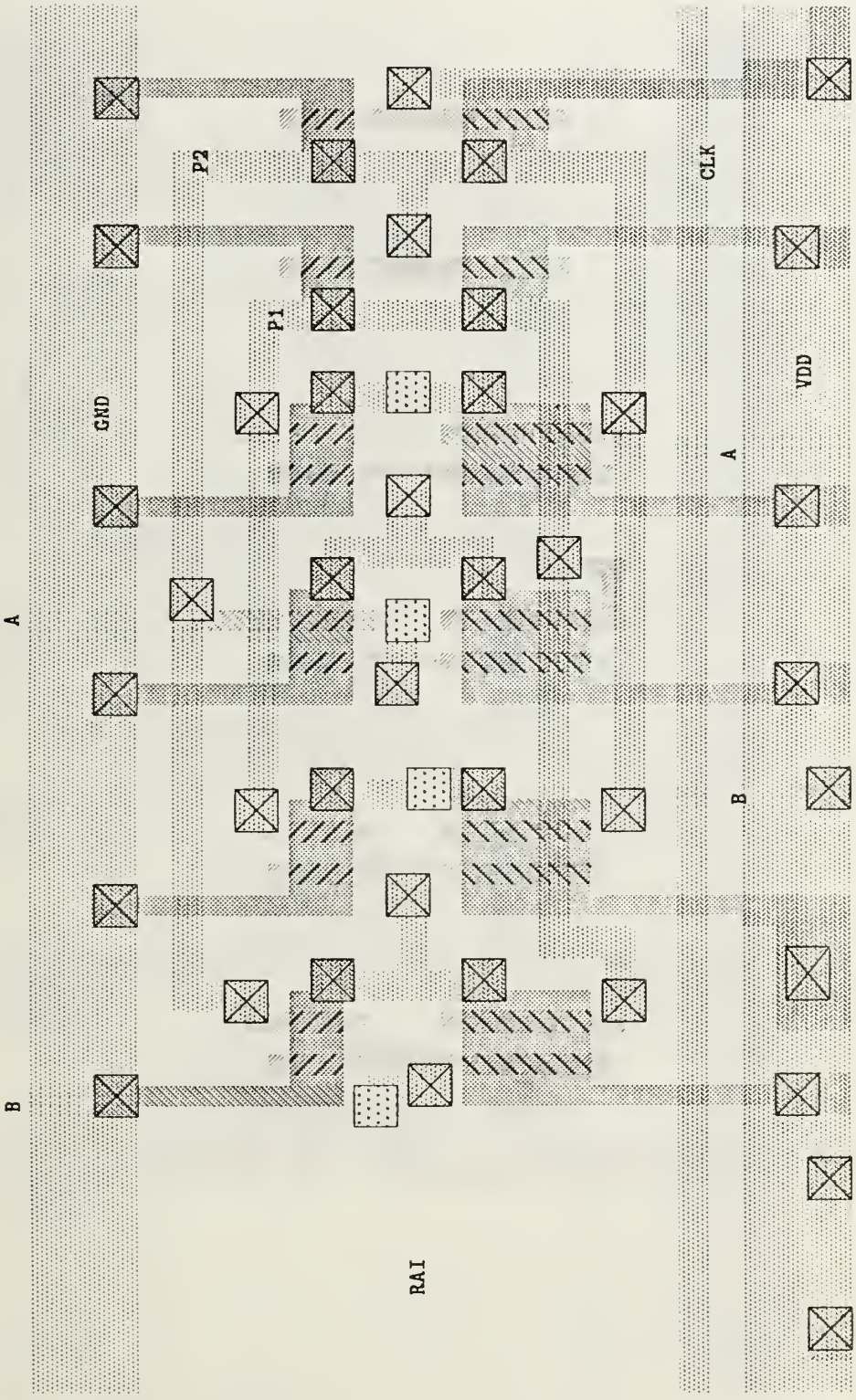


Figure 57. Cell RAI layout

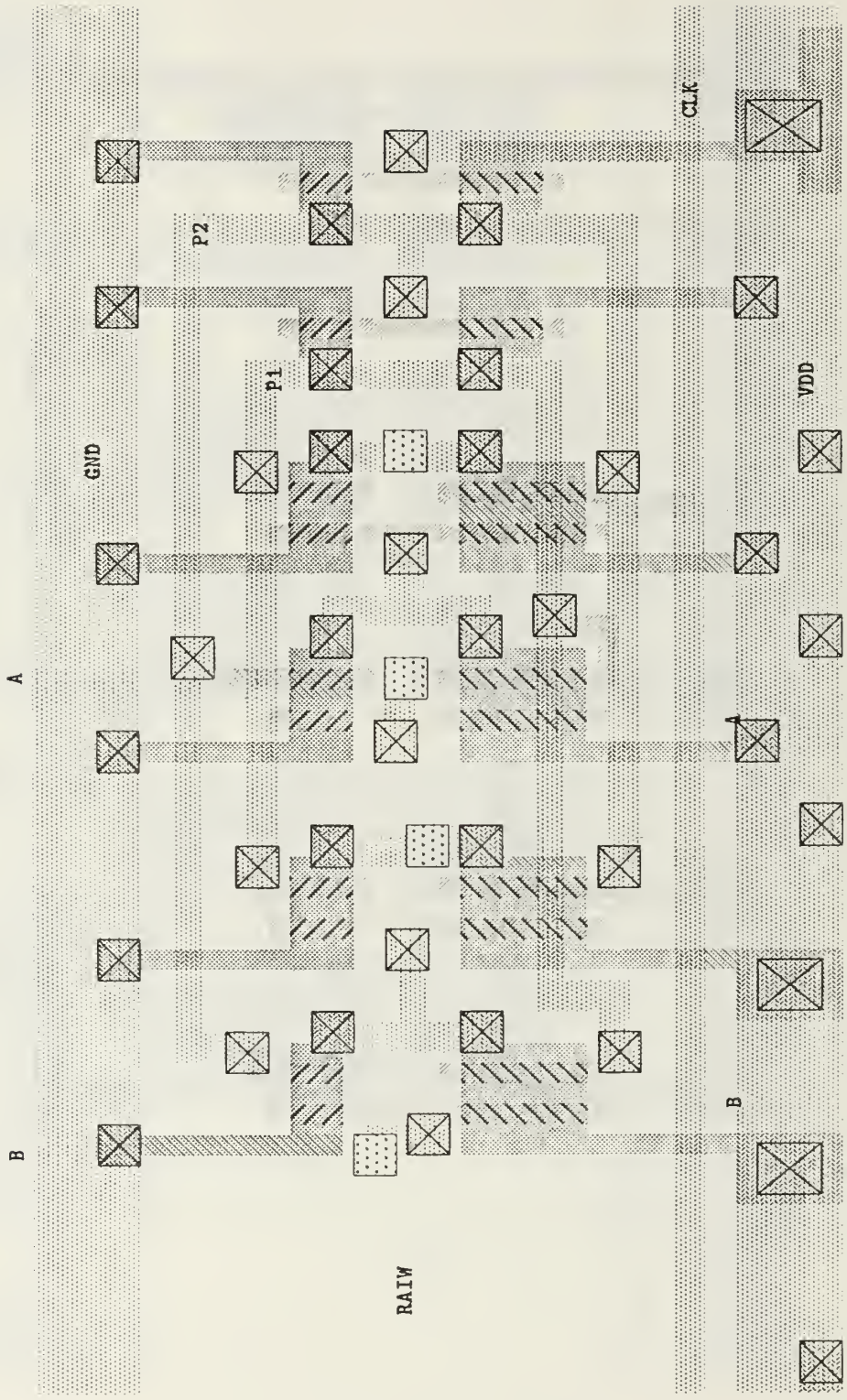


Figure 58. Cell RAIW layout

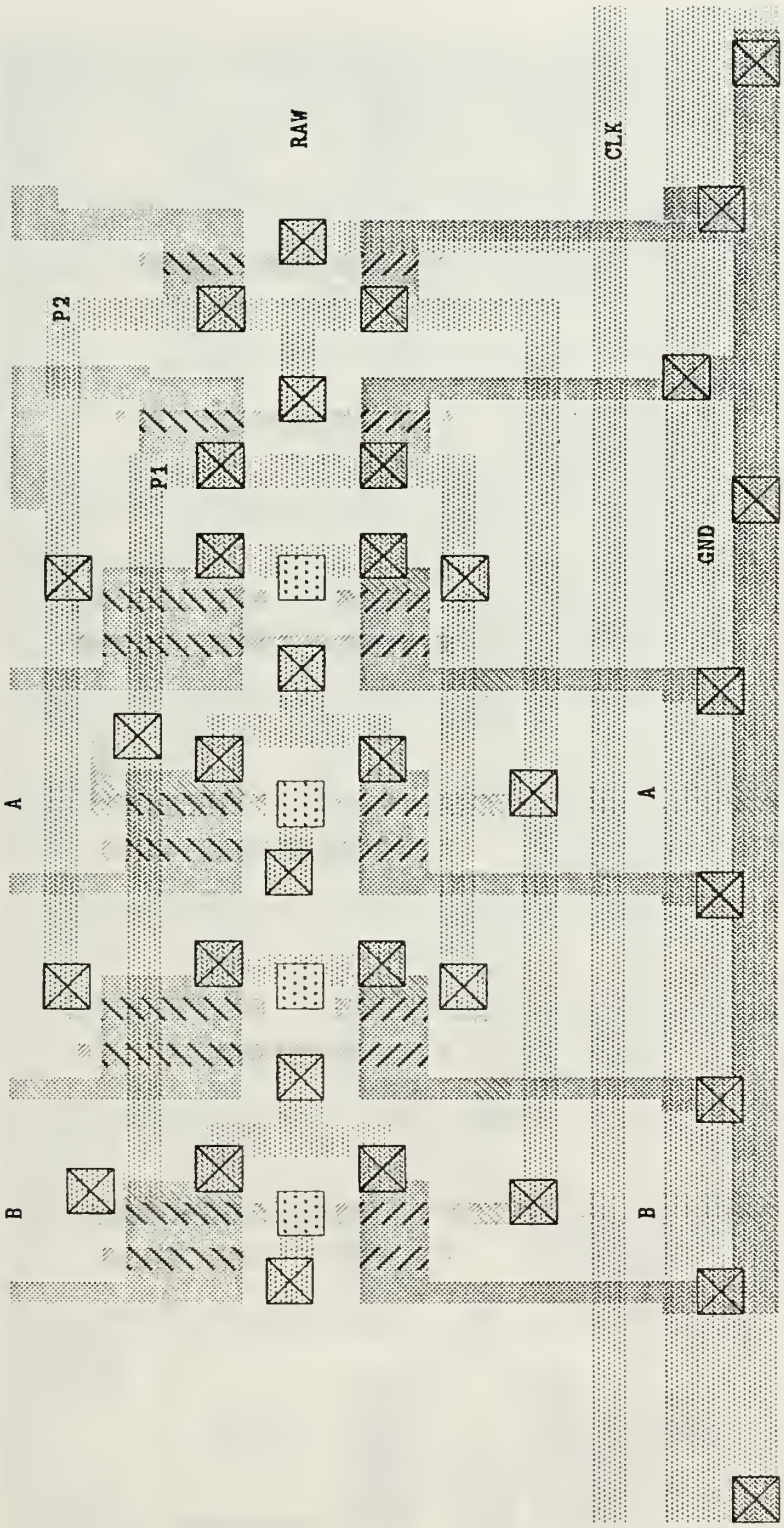


Figure 59. Cell RAW layout

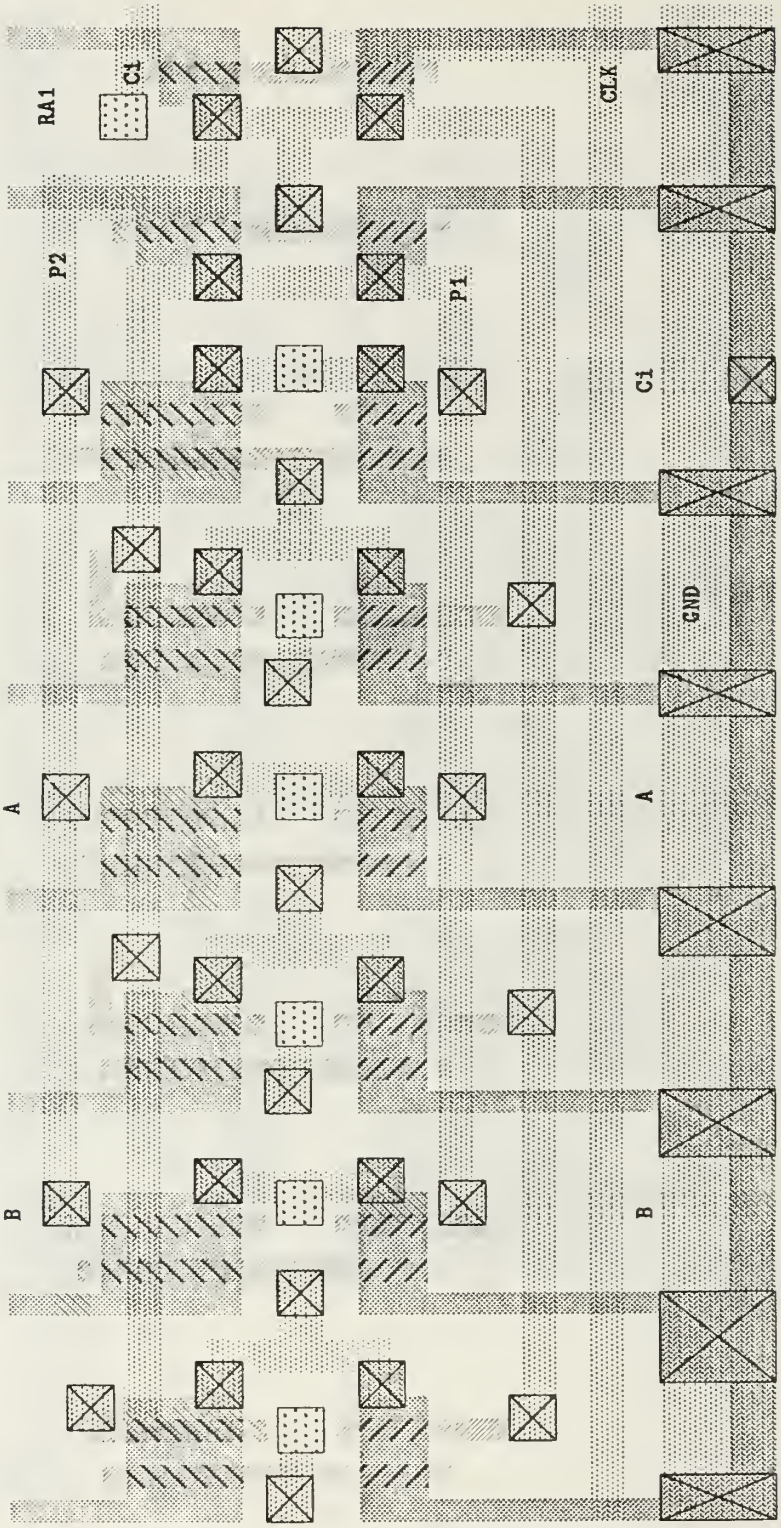


Figure 60. Cell RA1 layout

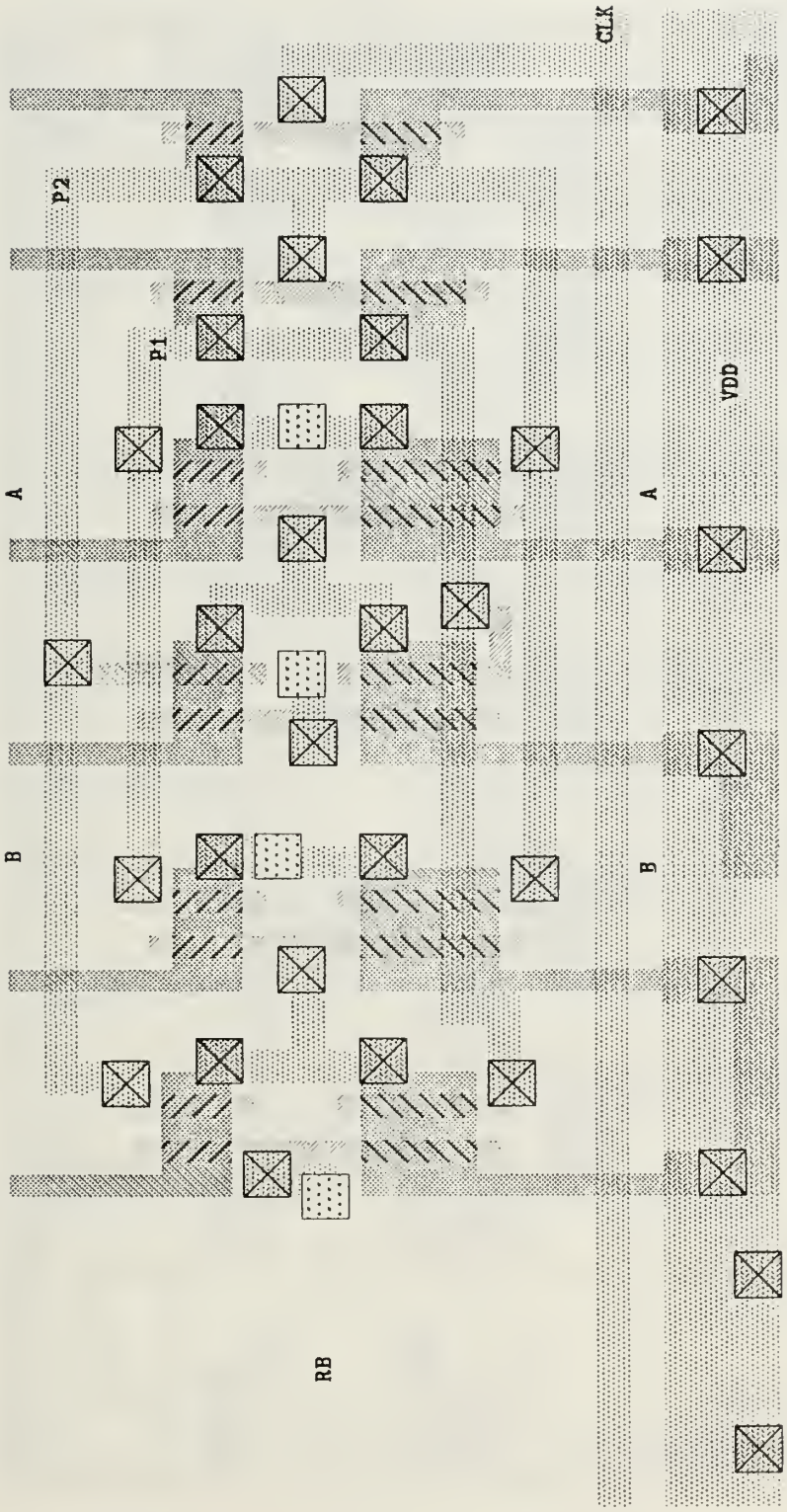


Figure 61. Cell RB layout

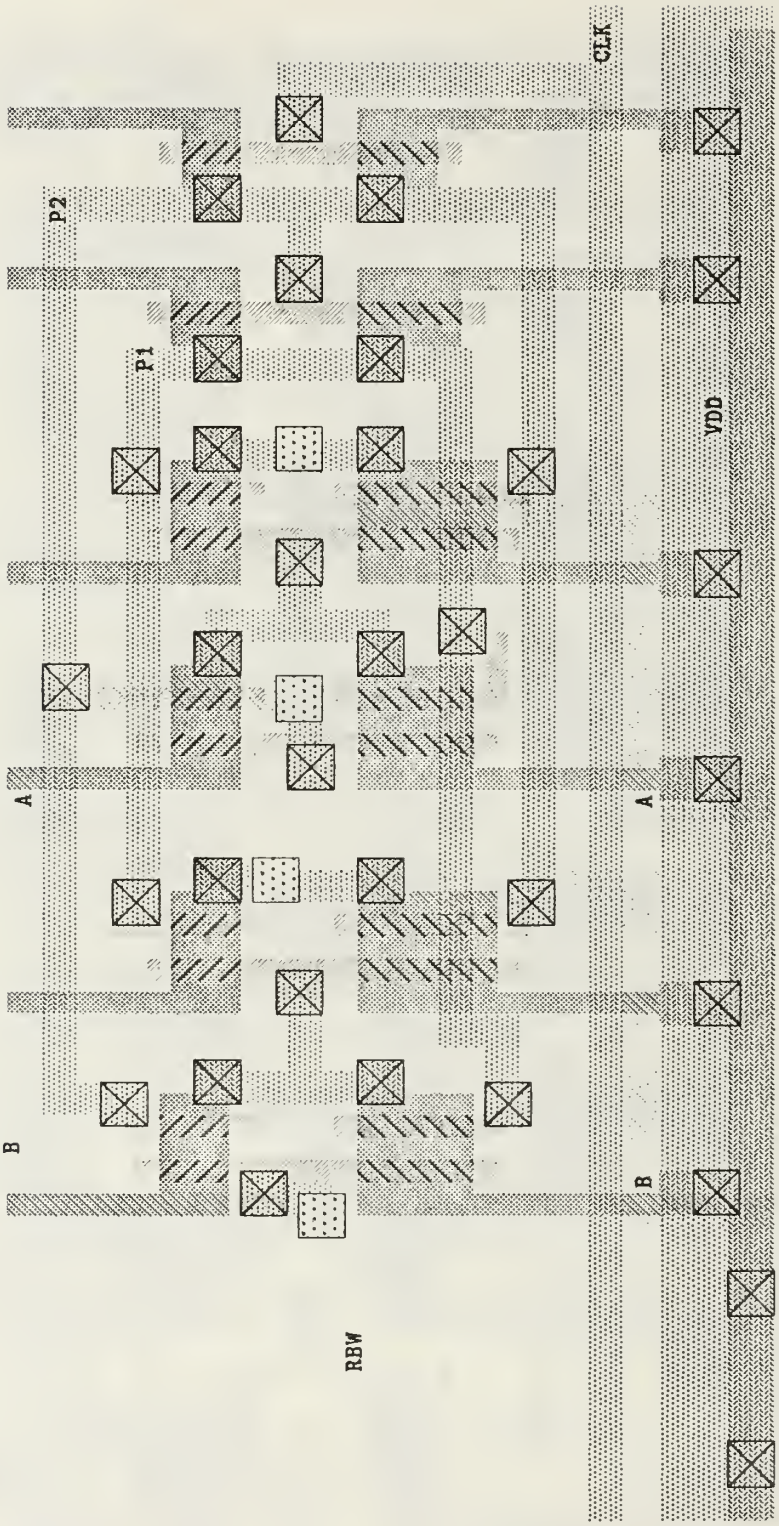


Figure 62. Cell RBW layout

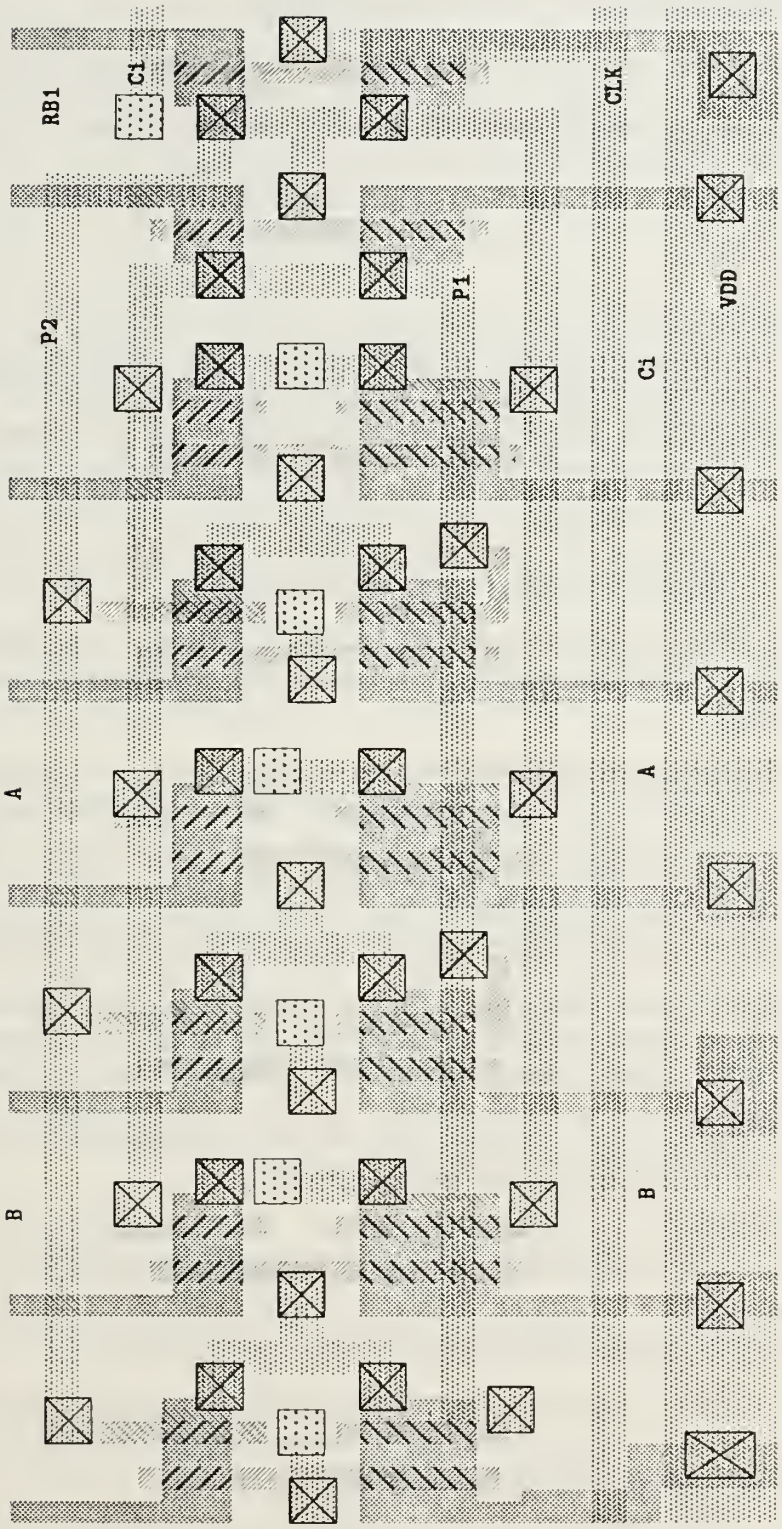


Figure 63. Cell RB1 layout

B. MULTIPLICATION FUNCTION CELLS.

The standard cells used to perform the multiplication function are listed below. An example arrangement of cells to perform a 16-bit mantissa multiply is shown in Figure 36 on page 64. The following list describes the cells and their functions:

Cell BMA: Figure 64 on page 108; a cell used to provide space for vertical VDD and GND bus lines. The cell also contains a latch to store the Y bit used on that level for the multiply process. The Y bit is sent on line Yp to the cell one clock cycle prior to use, stored, and then distributed to cells nearby. The control lines for the latch, P1 and P2, are driven by CMRC cells located above the BMA cells.

Cell BMB: Figure 65 on page 109; similar to cell BMA, but has a VDD (vice GND) bus running along its lower edge.

Cell CAE: Figure 66 on page 110; a "clock" cell that contains vertical GND and VDD busses. Also contains a clock inverter driven by the CLKin line.

Cell CMRA: Figure 67 on page 111; the uppermost clock cell in the multiplier. It contains an inverter that drives the CLKin line for all the clock cells in that column. The cell also contains vertical VDD and GND busses, an inverter to drive the CLK lines in nearby cells in the same row, and clock inverters that drive the latch control lines (P1 and P2) for the BMA and BMB cells located below the CMRA cells.

Cell CMRB: Figure 68 on page 112; a "three level" cell which provides space for vertical VDD and GND busses. It also contains clock inverters to provide CLK signals to nearby cells in the same row.

Cell CMRC: Figure 69 on page 113; similar to cell CMRB, but with the horizontal VDD and GND busses reversed.

Cell CN: Figure 70 on page 114; a clock cell that functions similarly to cell CAE, but with a VDD bus running along the bottom.

Cell CP: Figure 71 on page 115; a clock cell similar to cell CAE, but with additional lines which pass signals through the cell.

Cell CS: Figure 72 on page 116; a clock cell used in the "selection" row at the bottom of the multiplier. It contains eight lines necessary to pass the select control line signals through the cell.

Cell CYA: Figure 73 on page 117; a clock cell used in the top row to provide VDD, GND, and CLK distribution to the latches that store an distribute the Y mantissa bits.

Cell CYC: Figure 74 on page 118; similar to cell CYA, but with no GND bus at the top.

Cell FA: Figure 75 on page 119; a cell that shifts the Y mantissa bits to the right one place every multiplier level to provide for proper distribution of the Y bits. Also grounds the PRODUCT-IN (Pi) lines of the left-most column of multiplier cells.

Cell FB: Figure 76 on page 120; similar to cell FA, but has a GND bus running along its lower edge.

Cell FBA: Figure 77 on page 121; provides space for vertical VDD and GND busses running between CYC cells.

cell FBB: Figure 78 on page 122; similar to cell FBA, but with a VDD bus running along its bottom edge.

cell FYA: Figure 79 on page 123; similar in function to cell FA.

Cell FYB: Figure 80 on page 124; similar in function to cell FB.

Cell M1A: Figure 81 on page 125; a multiplier cell. The inputs are Pi, Ci, X, and Y; the outputs are Po and Co. Figure 24 on page 48 shows the logic functions of this cell.

Cell M1B: Figure 82 on page 126; similar in function to cell M1A, but with a VDD bus running along its lower edge.

Cell NA: Figure 83 on page 127; a three level cell. The top level functions as an full adder cell that adds the PRODUCT and CARRY bits from the bottom row of multiplier cells. The second row performs the alternate rounding function (see Figure 29 on page 52) on the product term one bit less in significance than the product term provided by the full adder level above. The center level also contains two latches to store the product (P) and alternate product (Pm) bits. The lower level of cell NA contains an OR gate used in a zero sensing scheme, two latches used to store P and Pm, and a pair of inverters used to drive the control lines for the latches used in the cell.

Cell NAE: Figure 84 on page 128; functions similarly to cell NA, but is designed to be the last logic cell in a pipeline stage. To Prevent the alternate rounding logic from being in the critical timing path, the cell latches the Pi and Pi-1 bits before performing logic functions.

Cell NA1: Figure 85 on page 129; functions similarly to cell NA. Designed to be the first logic cell in a pipeline stage.

Cell NB: Figure 86 on page 130; functions the same as cell NA, but with VDD and GND bus locations switched.

Cell NBE: Figure 87 on page 131; functions the same as cell NAE, but with GND and VDD bus locations similar to cell NB.

Cell NBF: Figure 88 on page 132; designed to be the last cell in the ripple adder at the bottom to the multiplier. Unlike other NB type cells, no adder cell is needed in the top level because the Ci and Pi bits will always be zero. Cell NBF performs the last alternate rounding process, and contains latches, and control inverters to store appropriate bits.

Cell NBS: Figure 89 on page 133; designed as the first cell in the ripple add process at the end of the multiply. Similar in function and layout to cell NB, it contains an additional line to obtain and store the product term needed to start the alternate rounding process.

Cell NB1: Figure 90 on page 134; performs the same function as cell NA1, but with the VDD and GND bus layouts of cell NB.

Cell REA: Figure 91 on page 135; a latch cell that contains three latches and the latch control inverters driven by the CLK signal line.

Cell REB: Figure 92 on page 136; a latch cell with two latches and control inverters.

Cell REBP: Figure 93 on page 137; a latch cell similar in function to cell REA.

Cell REBS: Figure 94 on page 138; a latch cell similar to cell REA.

Cell RED: Figure 95 on page 139; a latch cell similar to cell REA, but with a GND bus along its lower edge.

Cell REE: Figure 96 on page 140; a latch cell similar to cell RED, but with only two latches.

Cell REEP: Figure 97 on page 141; a latch cell similar in function to cell RED.

Cell REES: Figure 98 on page 142; a latch cell similar in function to cell RED.

Cell RM1A: Figure 99 on page 143; a latch cell used at the top of the multiplier array. Cell RM1A initially stores the X bits for two clock cycles prior to the start of the multiply process. Cell RM1A also grounds the Pi and Ci lines of the upper row multiplier cells.

Cell RM1B: Figure 100 on page 144; a latch cell not needed if an odd number of multiplier cells are used in a pipeline stage. If an even number is used, cell RM1B is "alternated" with cell RM1C in the pipeline latch stages.

Cell RM1C: Figure 101 on page 145; a latch cell that stores the Pi, Ci, and X values between pipeline logic stages.

Cell SFM: Figure 102 on page 146; the selection cell used at the bottom of the multiplier to select one of four values. Requires four control signals and their complements. Only one control signal can be allowed to be active at any time. Cell SFM also contains a latch to store the selected value, and the control inverters necessary to operate the latch.

Cell YA: Figure 103 on page 147; a latch cell that stores four bits of the Y mantissa prior to use.

Cell YAA: Figure 104 on page 148; a cell made up of a GND bus used to provide for proper operation of cell YA when it is the topmost cell.

Cell YMC: Figure 105 on page 149; a latch cell similar in function to a YA and YAA pair.

Cell YRA: Figure 106 on page 150; a latch cell similar in function to a cell YA and YAA pair.

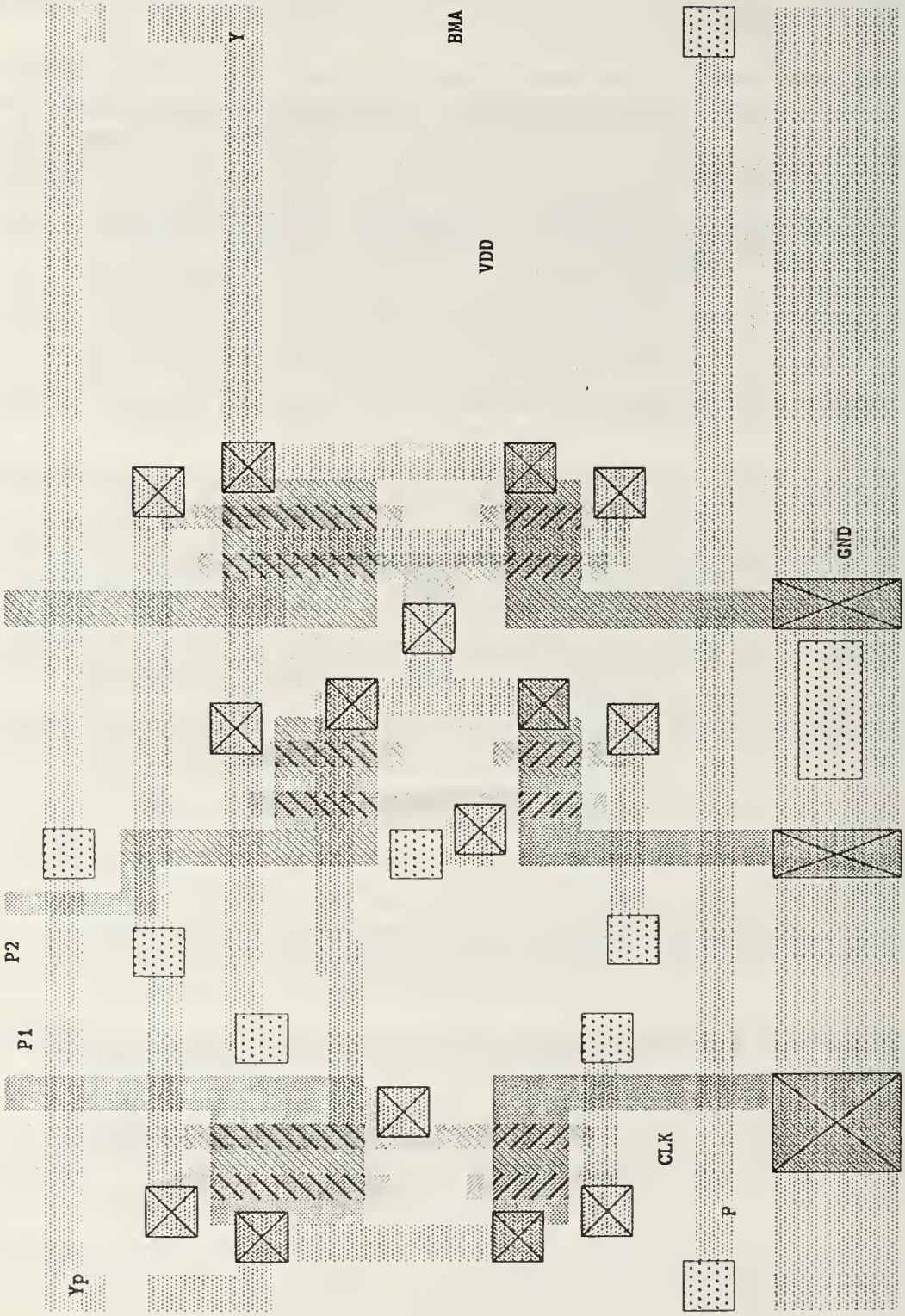


Figure 64. Cell BMA layout

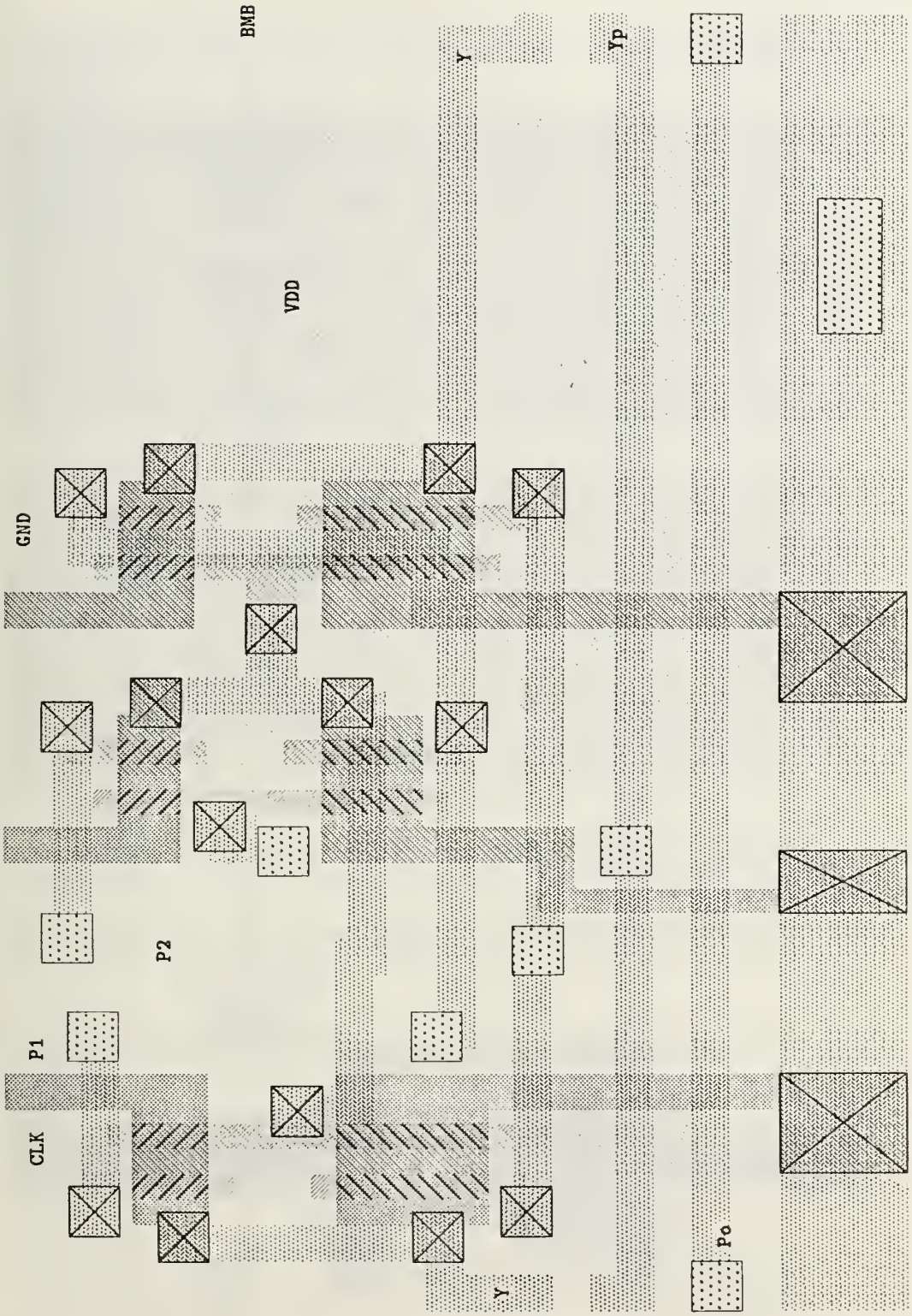


Figure 65. Cell BMB layout

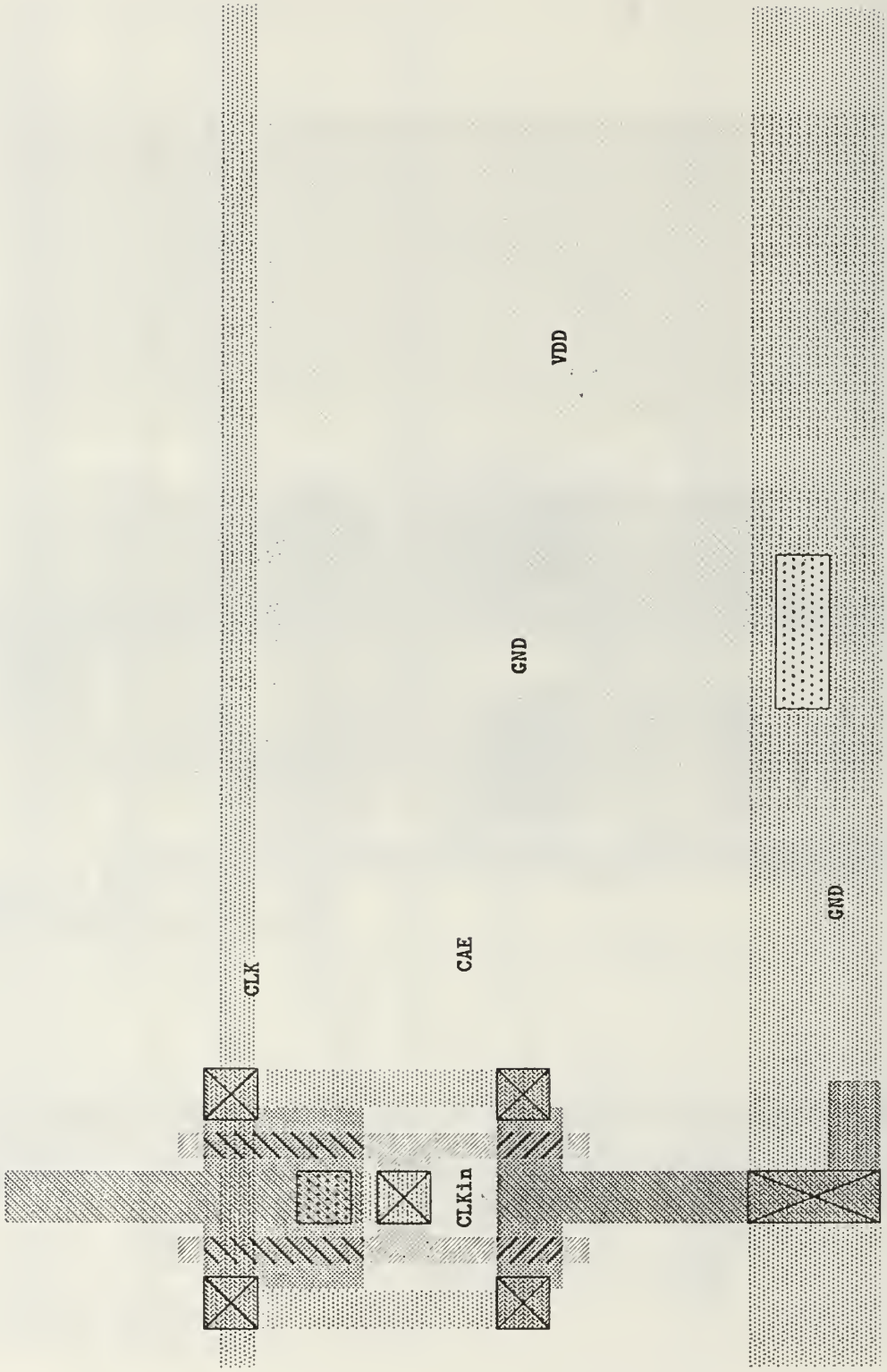


Figure 66. Cell CAE layout

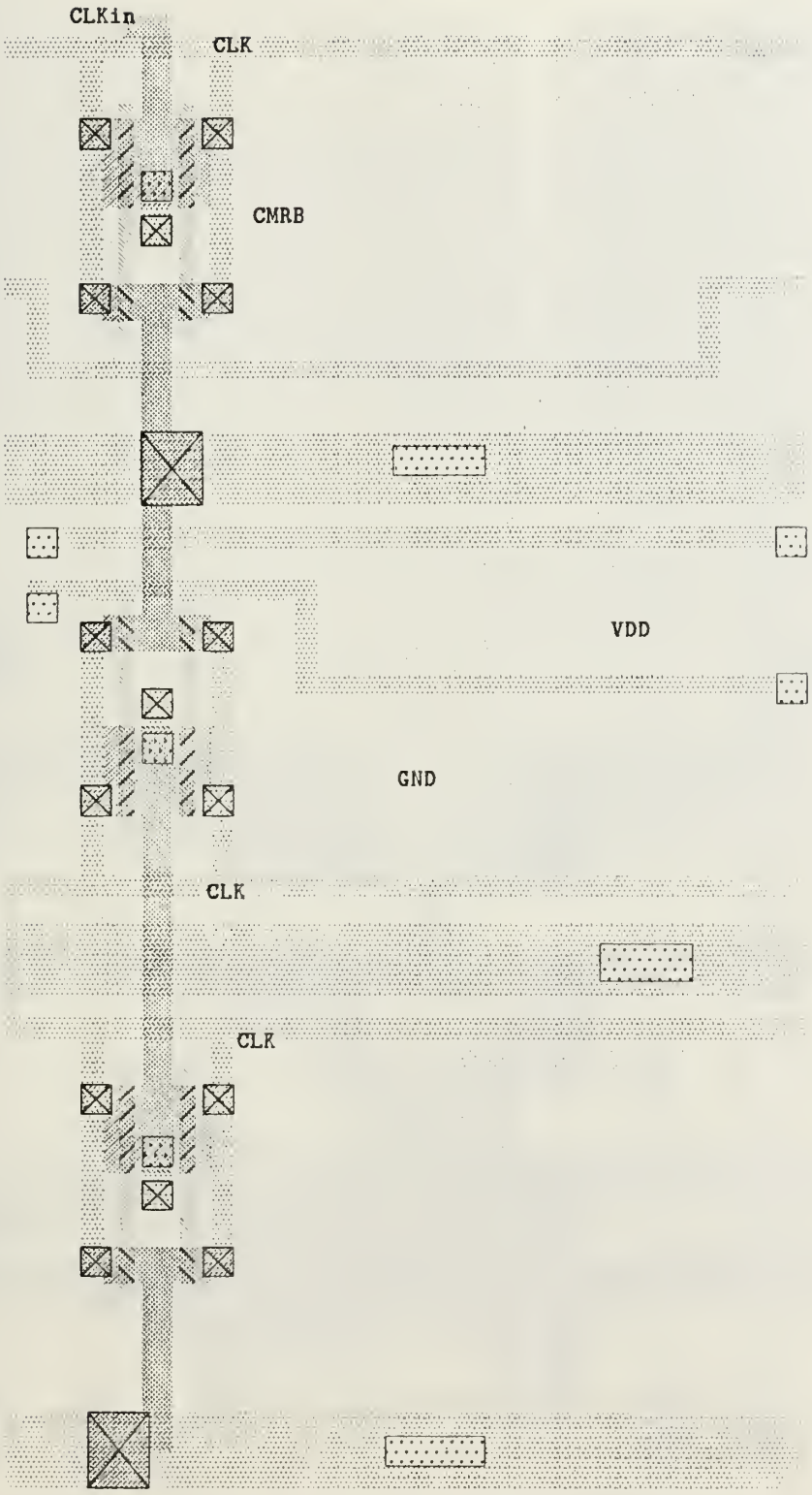


Figure 68. Cell CMRB layout

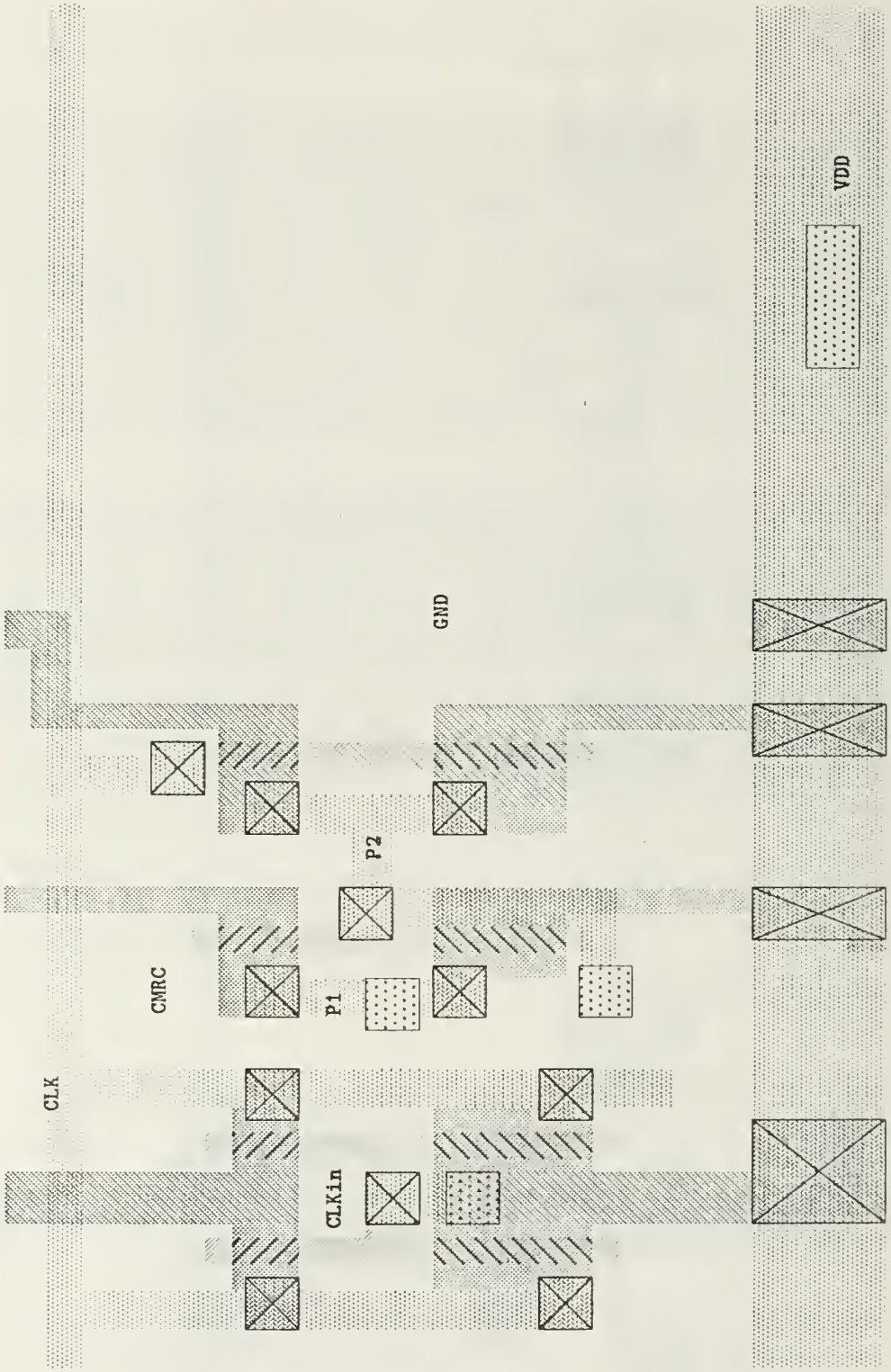


Figure 69. Cell CMRC layout

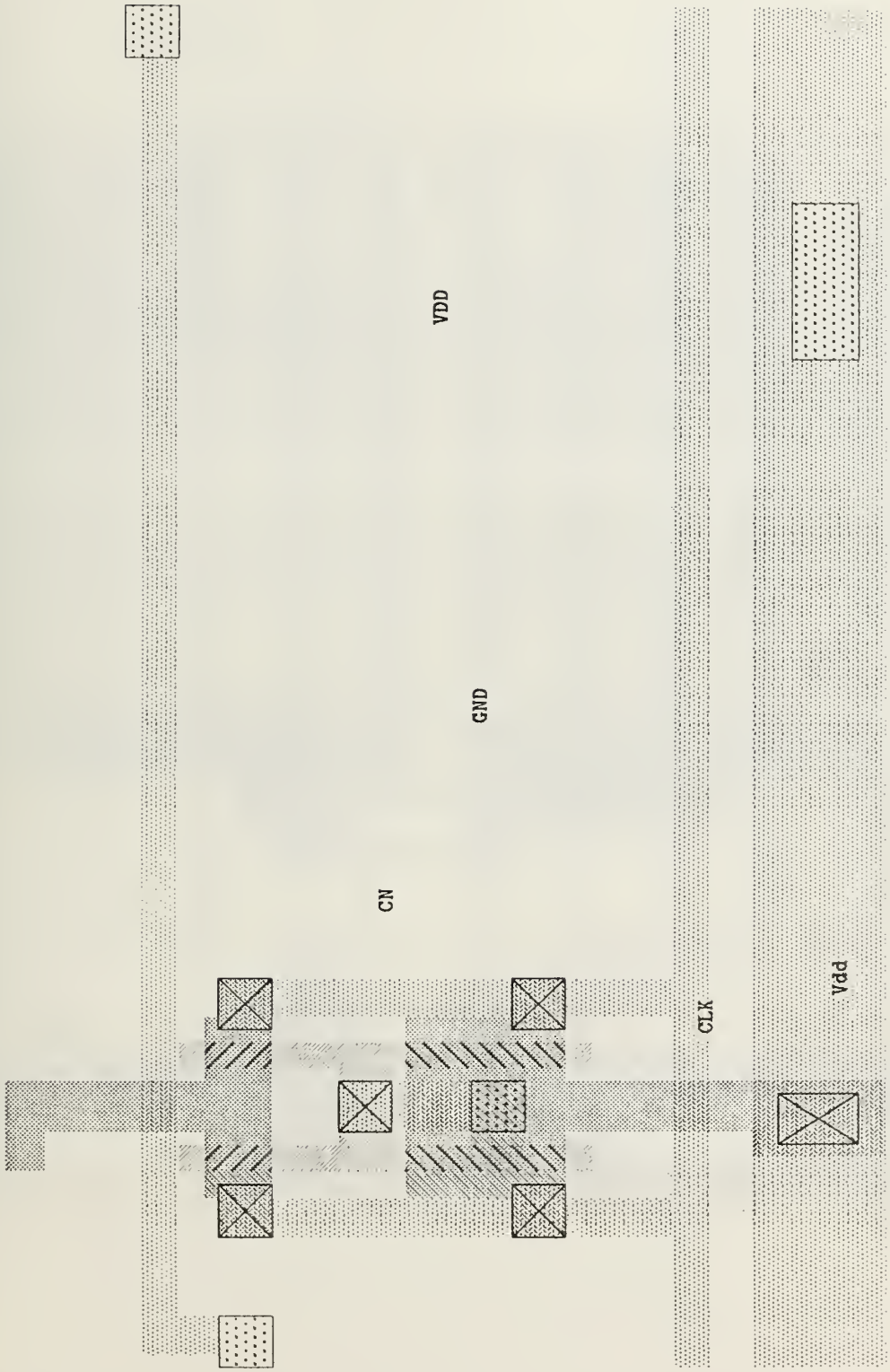


Figure 70. Cell CN layout

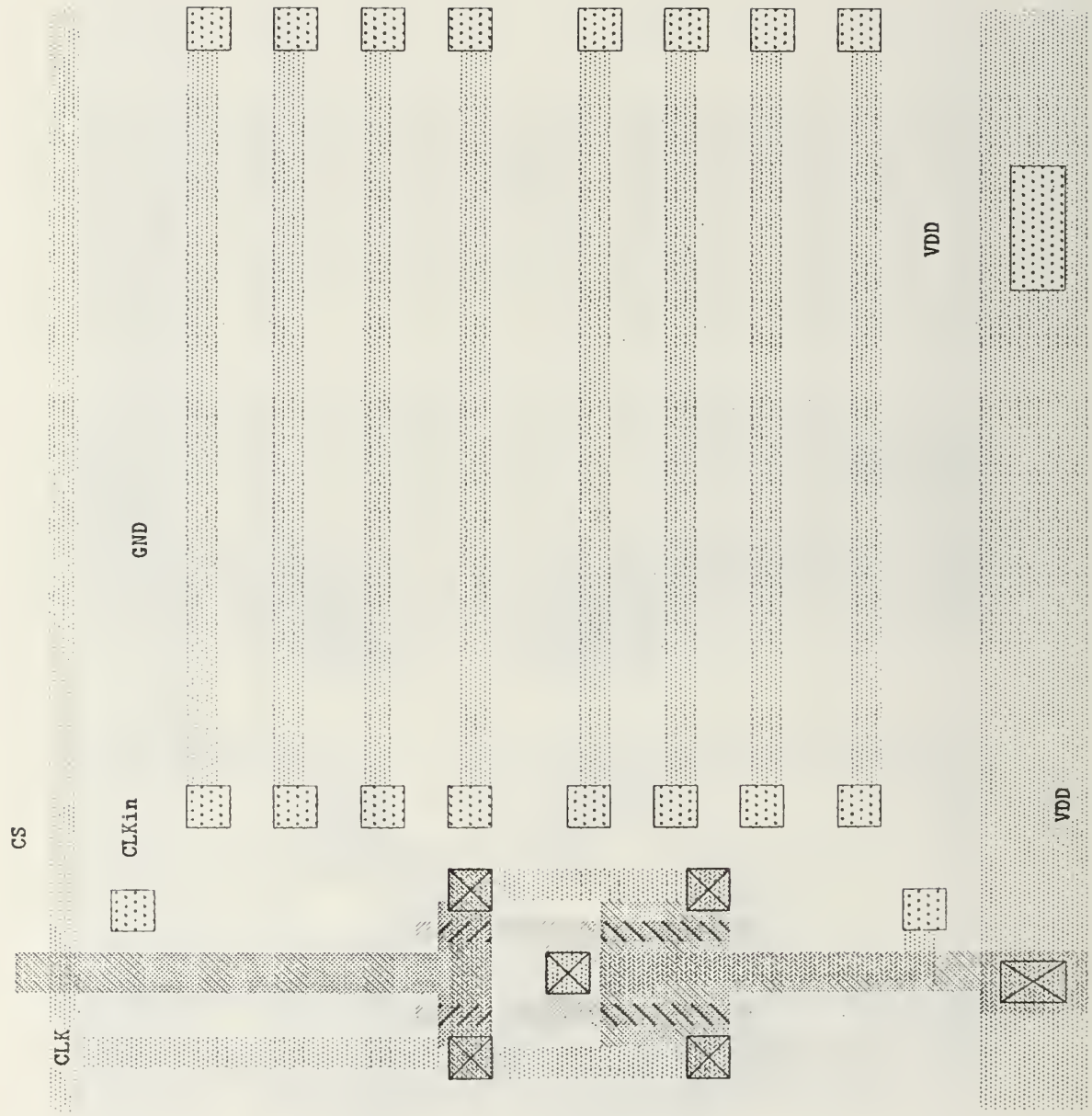


Figure 72. Cell CS layout

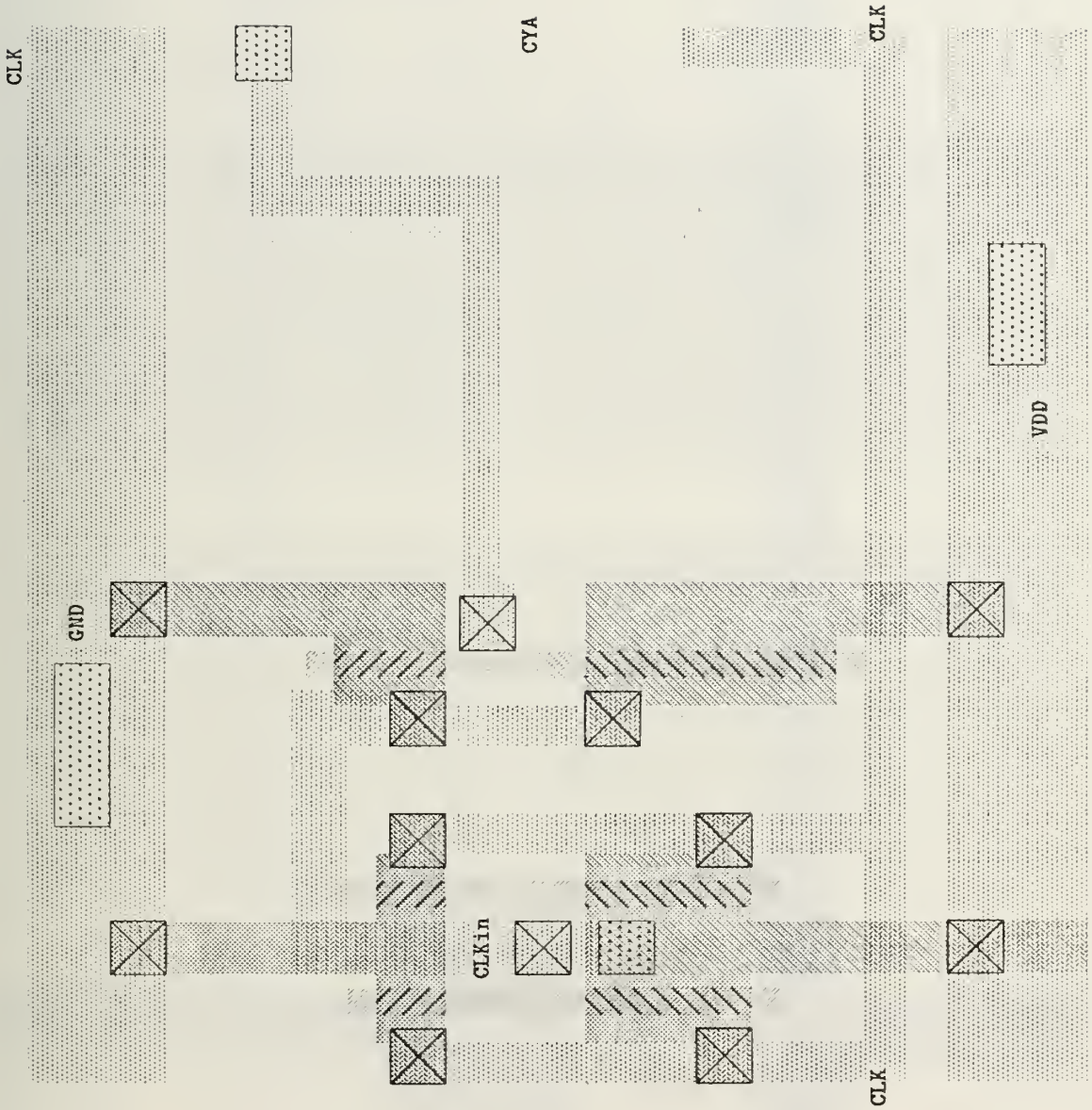


Figure 73. Cell CYA layout

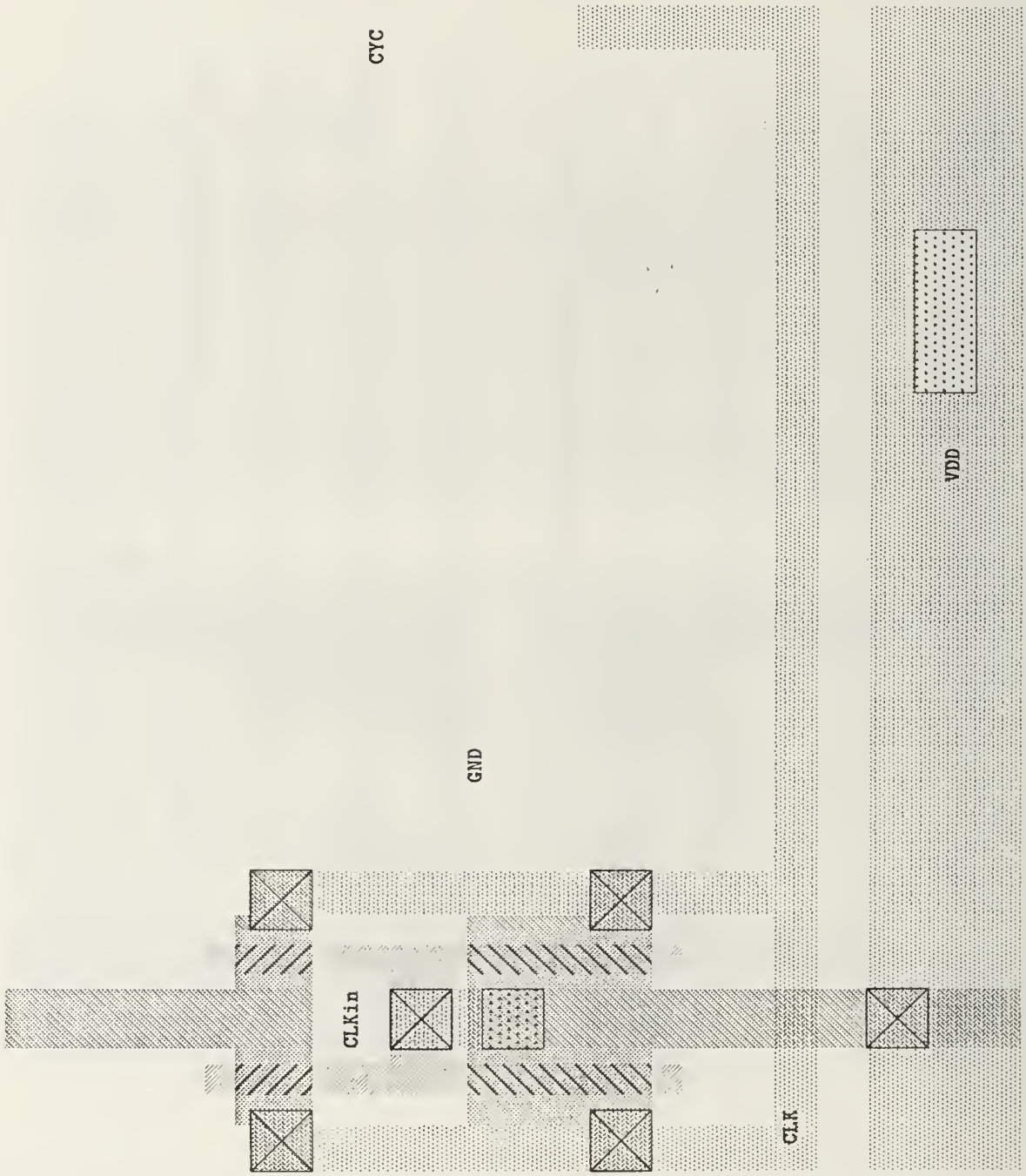


Figure 74. Cell CYC layout

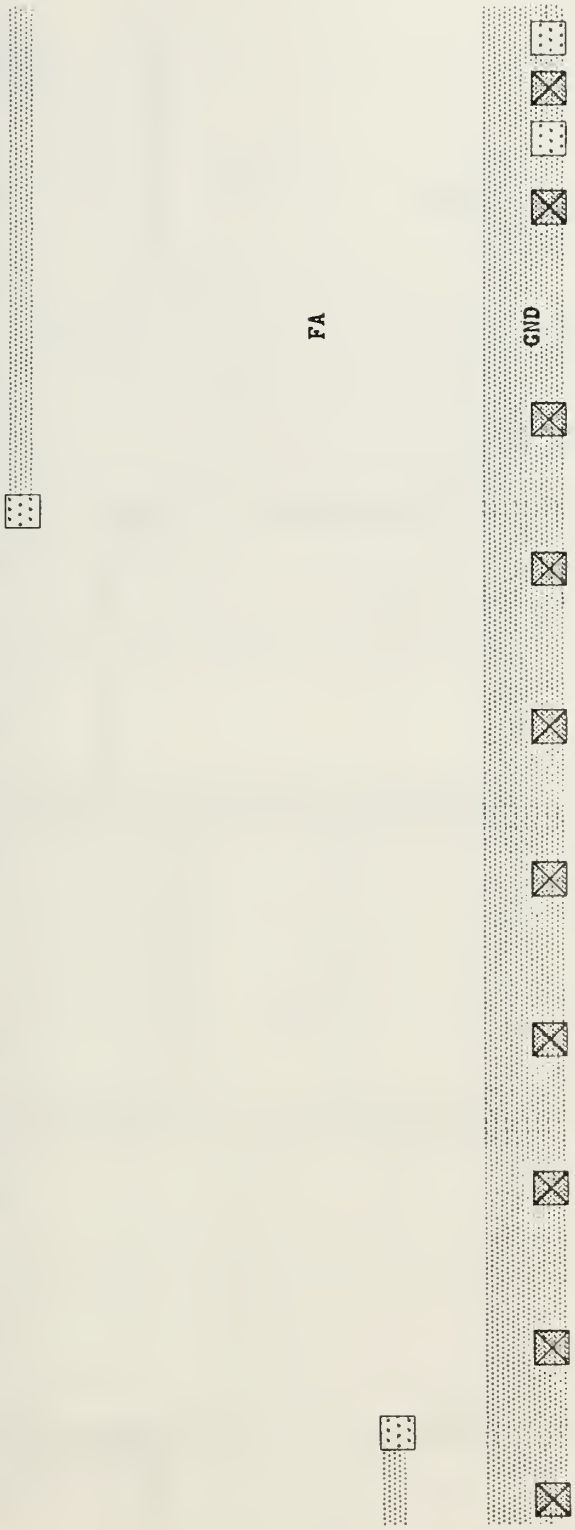


Figure 75. Cell FA layout

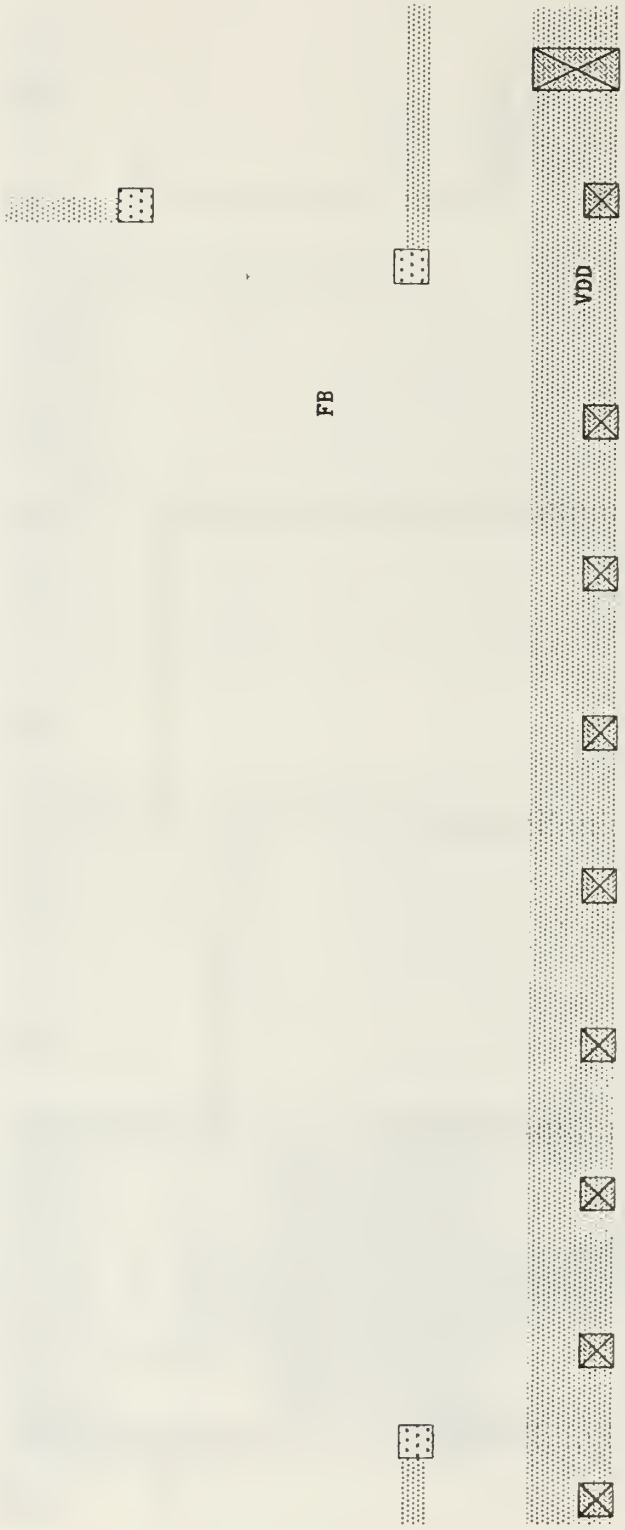


Figure 76. Cell FB layout

Figure 77. Cell FBA layout

Figure 78. Cell FBB layout

Figure 79. Cell FYA layout

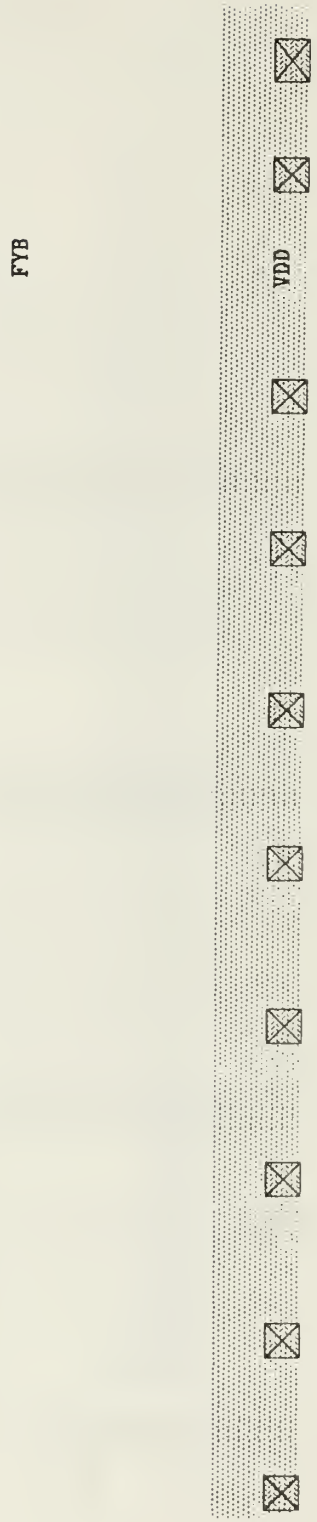


Figure 80. Cell FYB layout

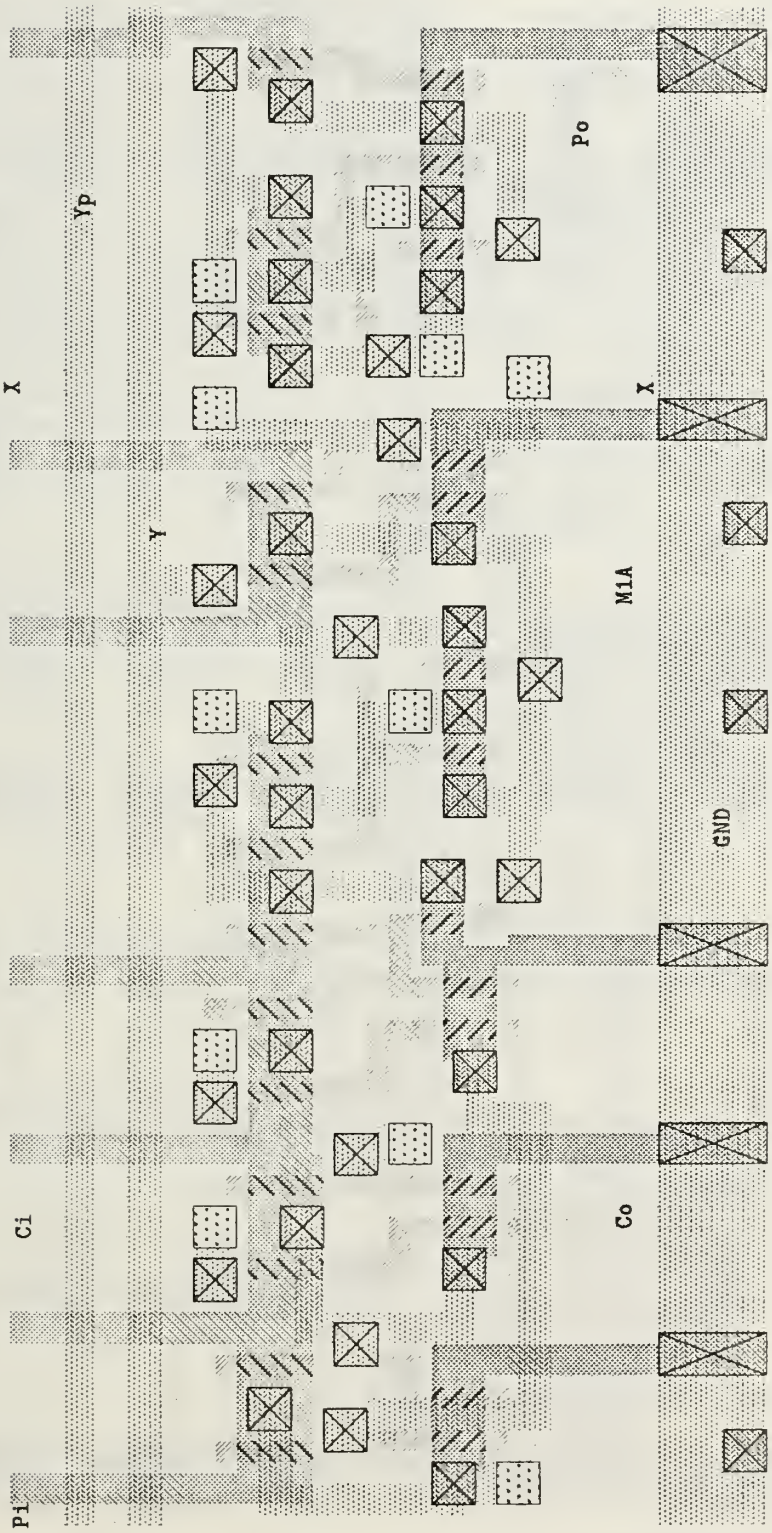


Figure 81. Cell M1A layout

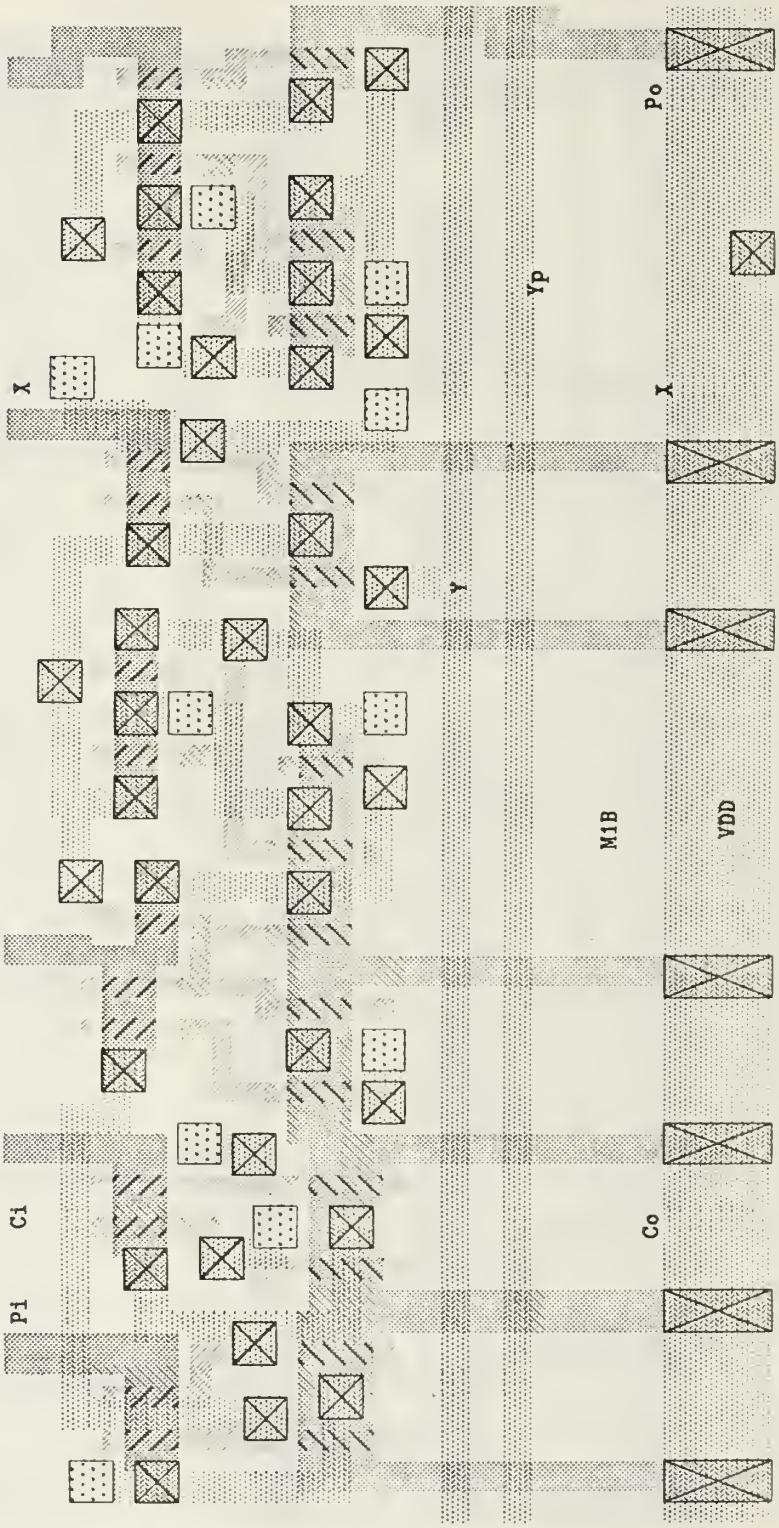


Figure 82. Cell M1B layout

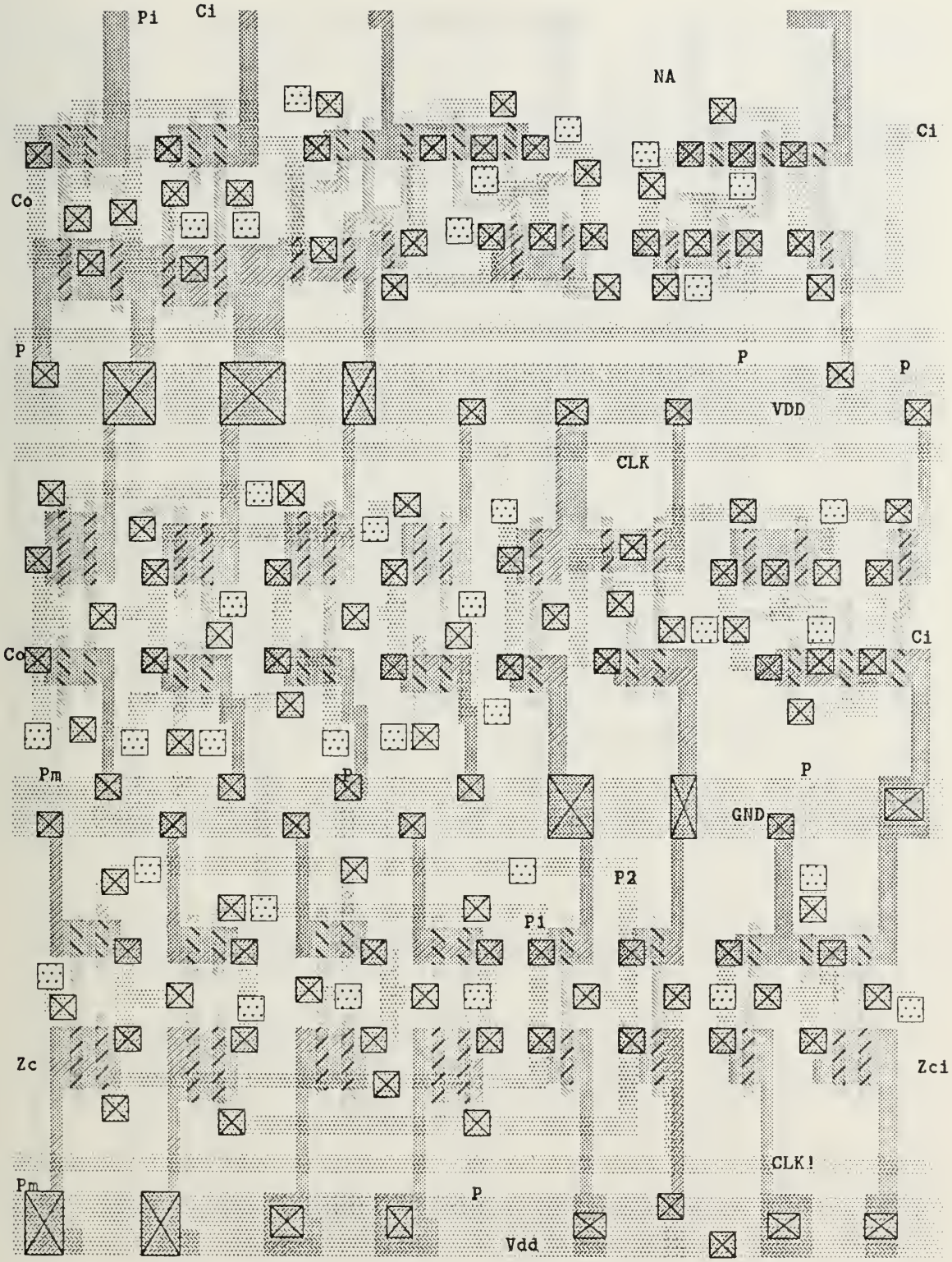


Figure 83. Cell NA layout

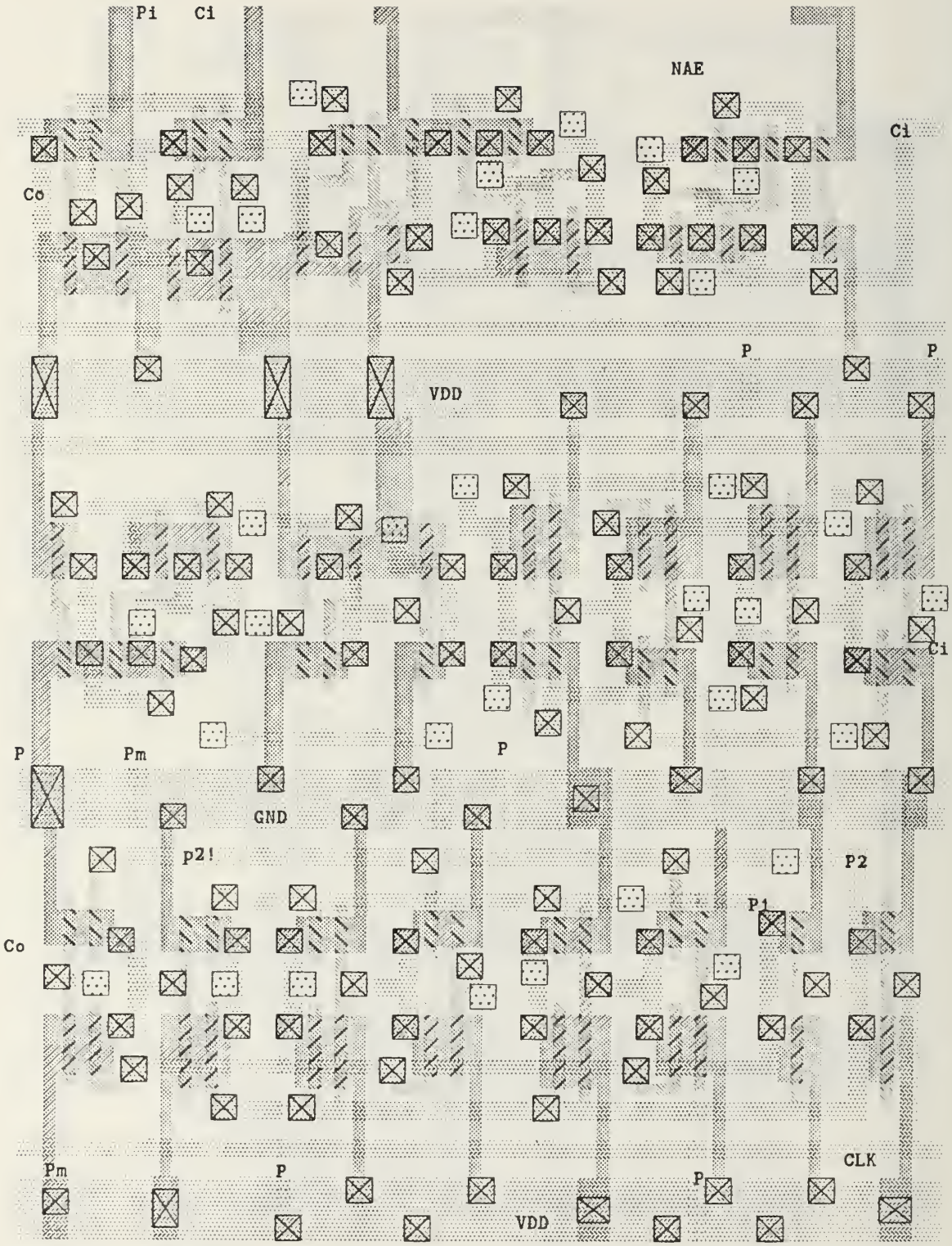


Figure 84. Cell NAE layout

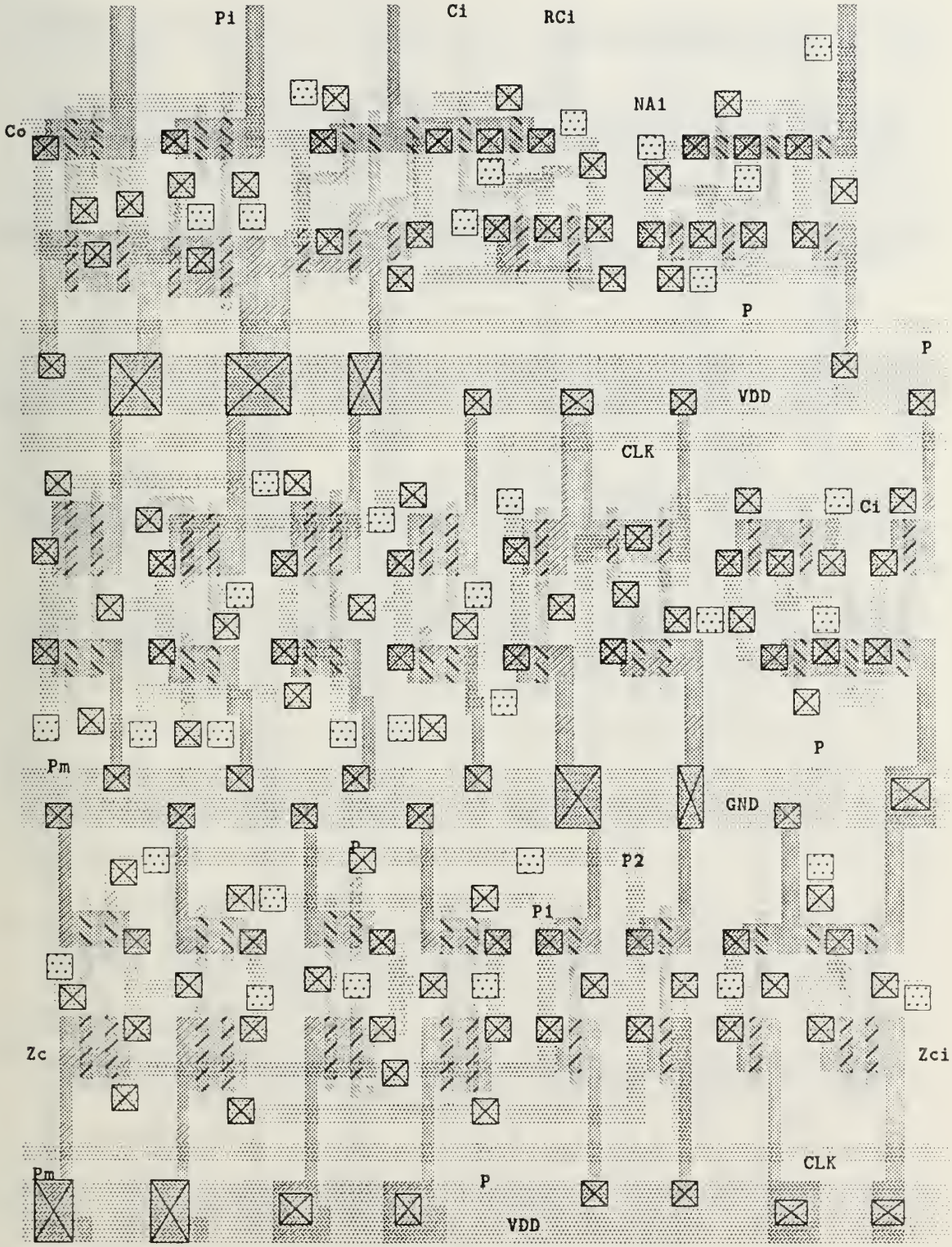


Figure 85. Cell NA1 layout

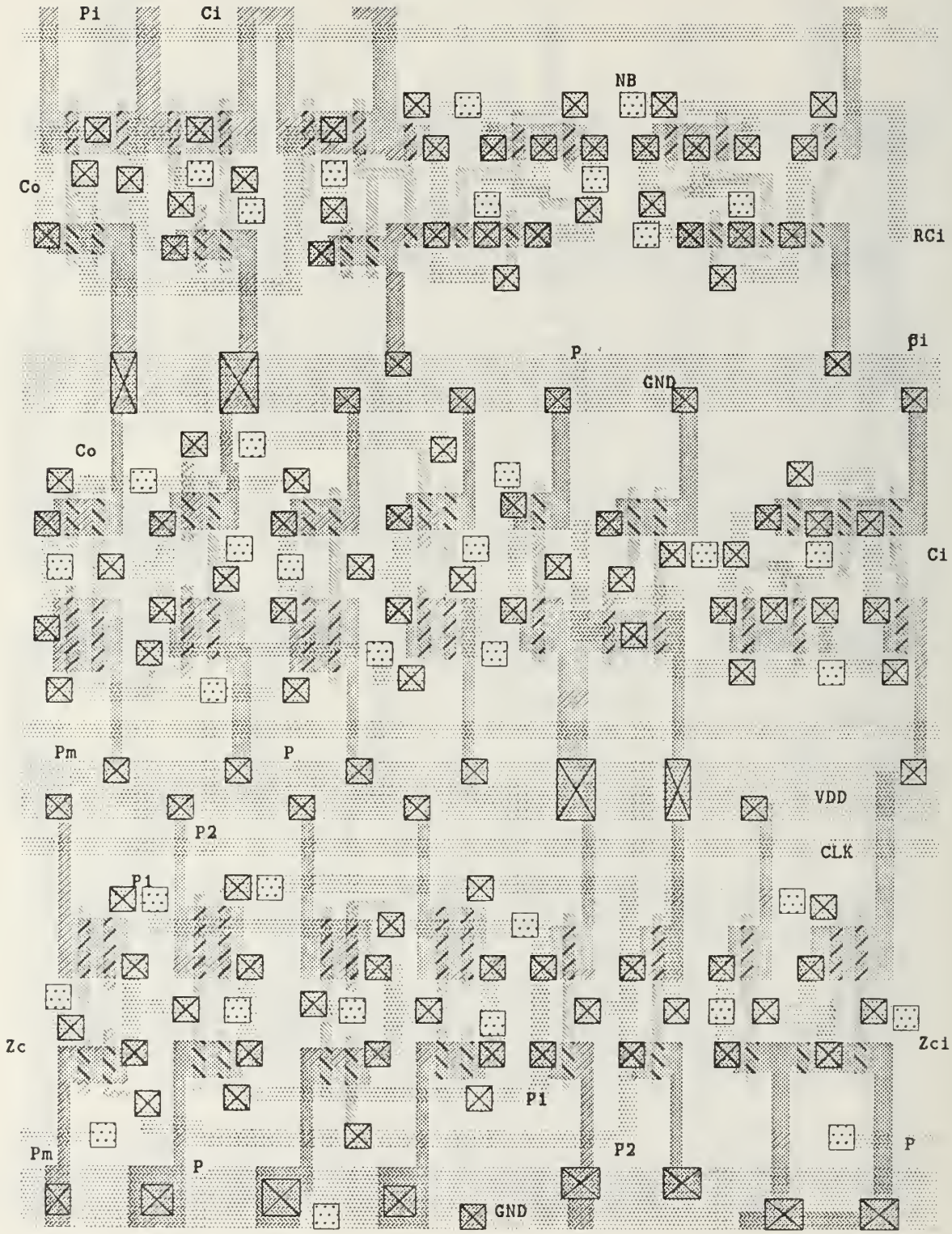


Figure 86. Cell NB layout

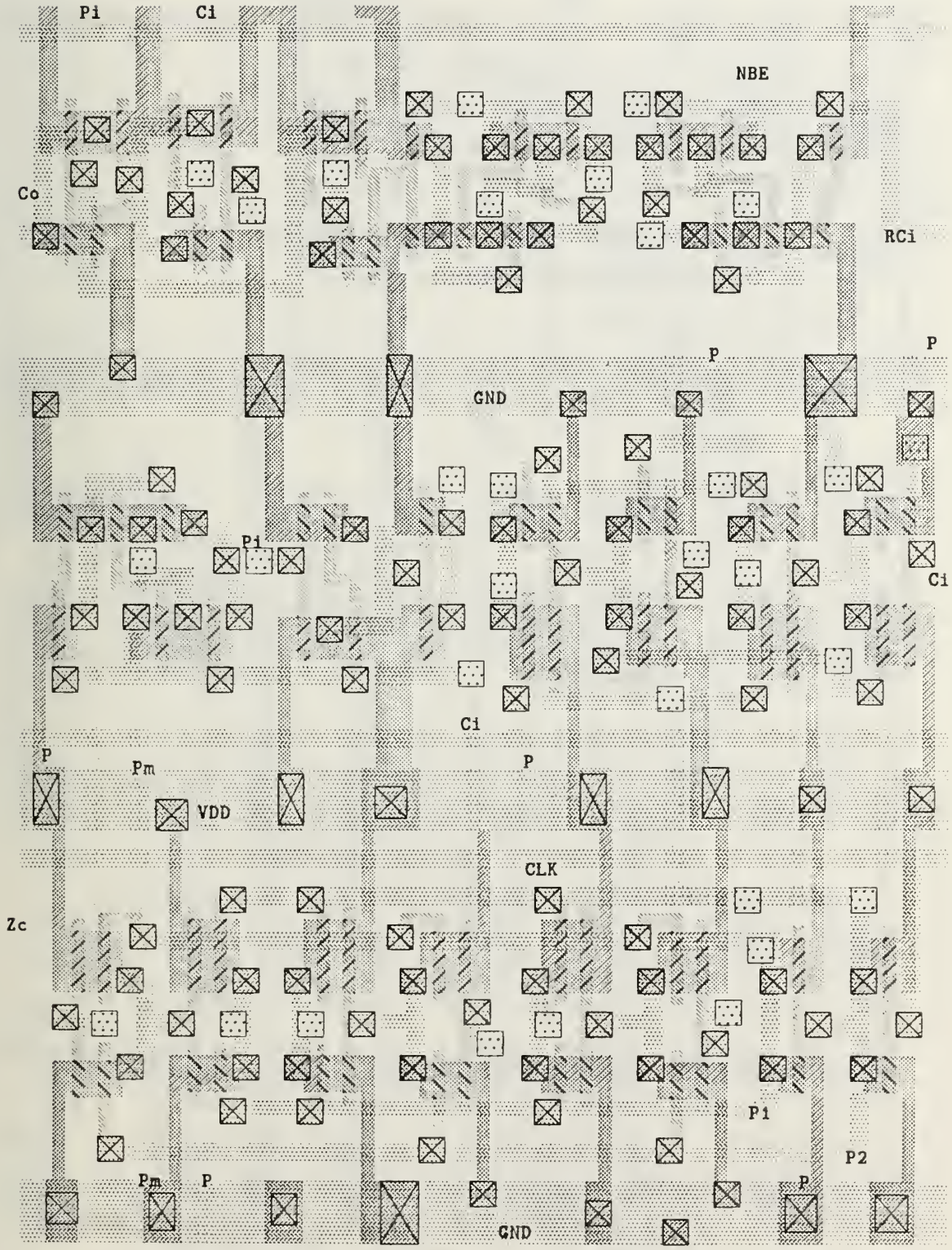


Figure 87. Cell NBE layout

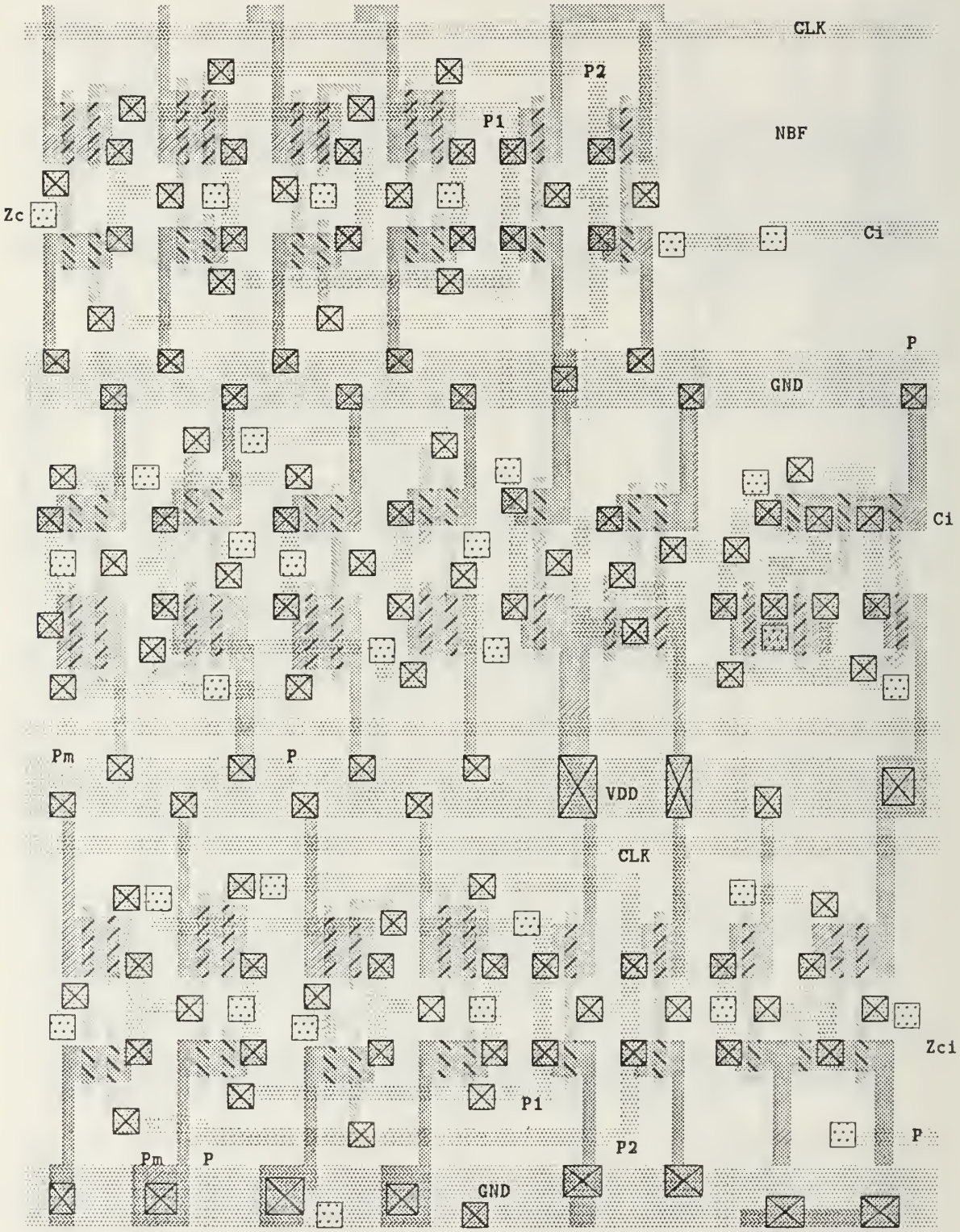


Figure 88. Cell NBF layout

P

NBS

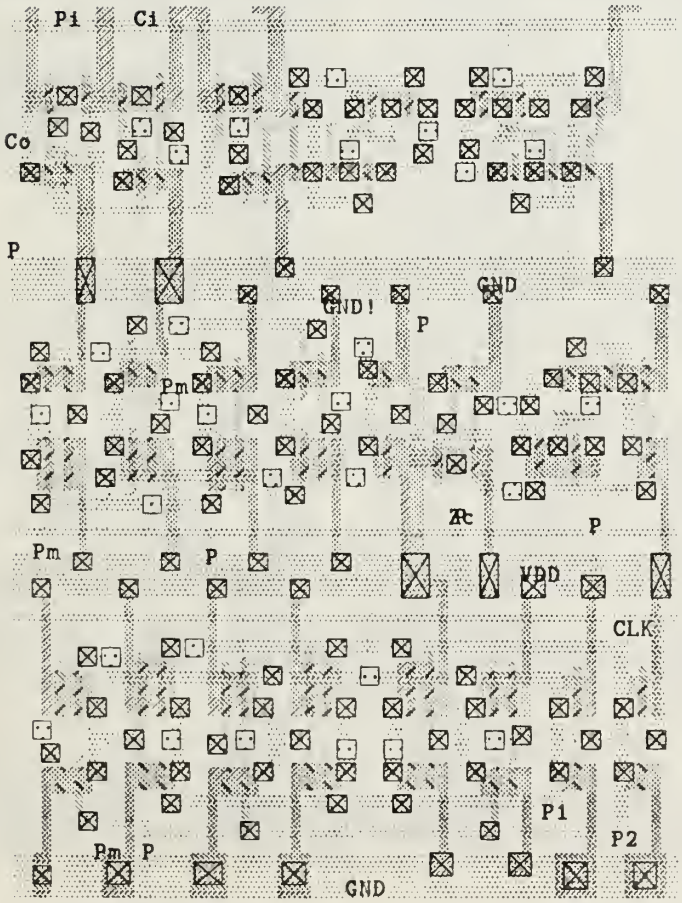


Figure 89. Cell NBS layout

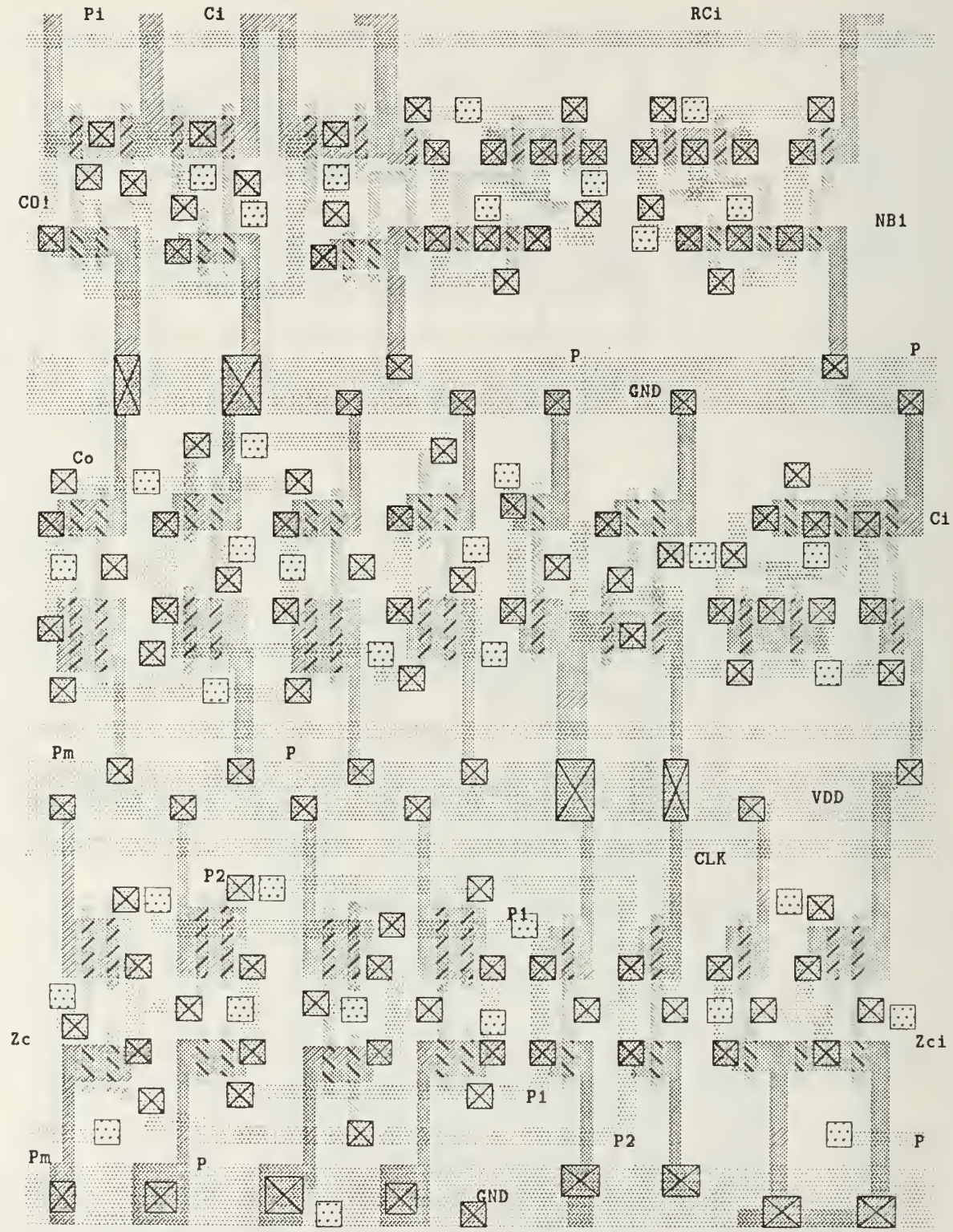


Figure 90. Cell NB1 layout

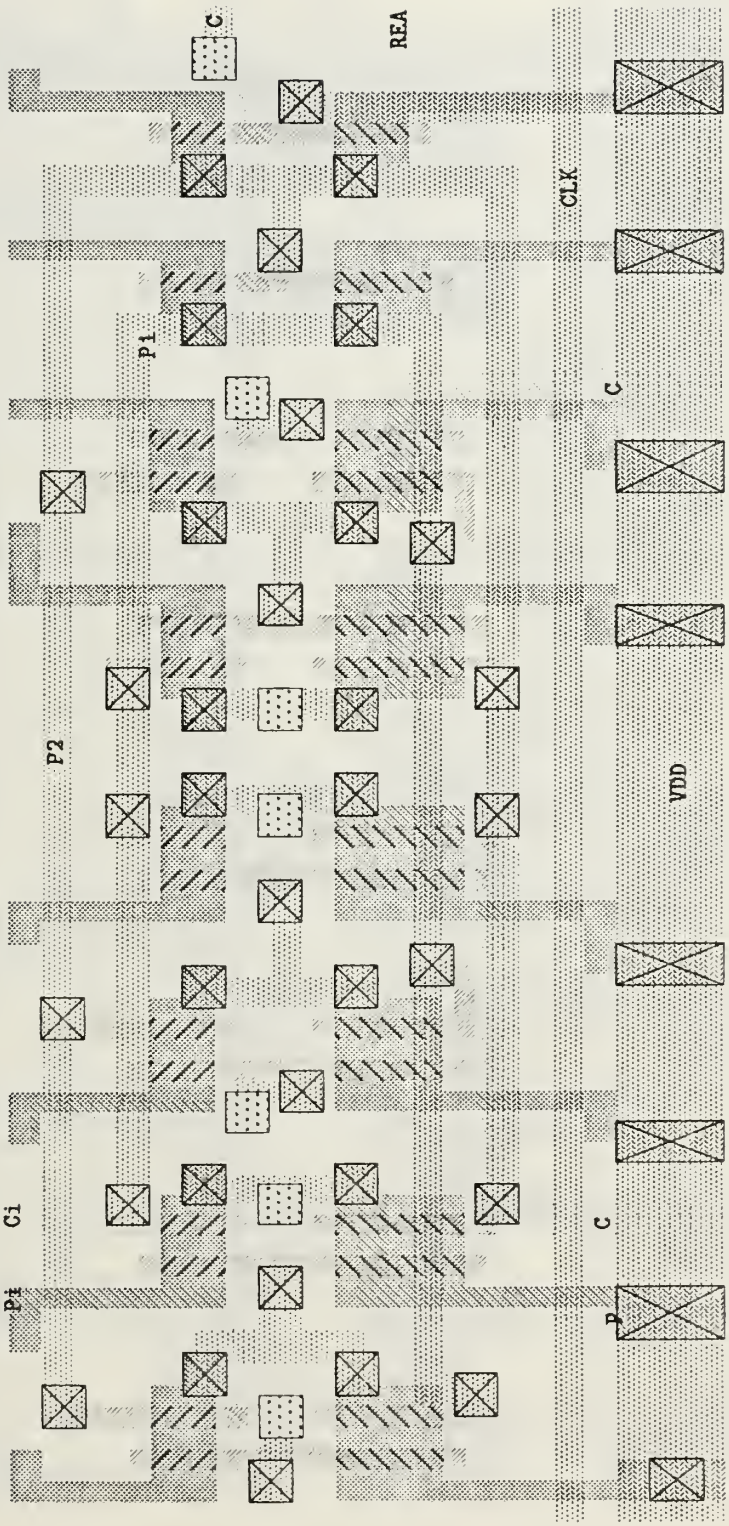


Figure 91. Cell REA layout

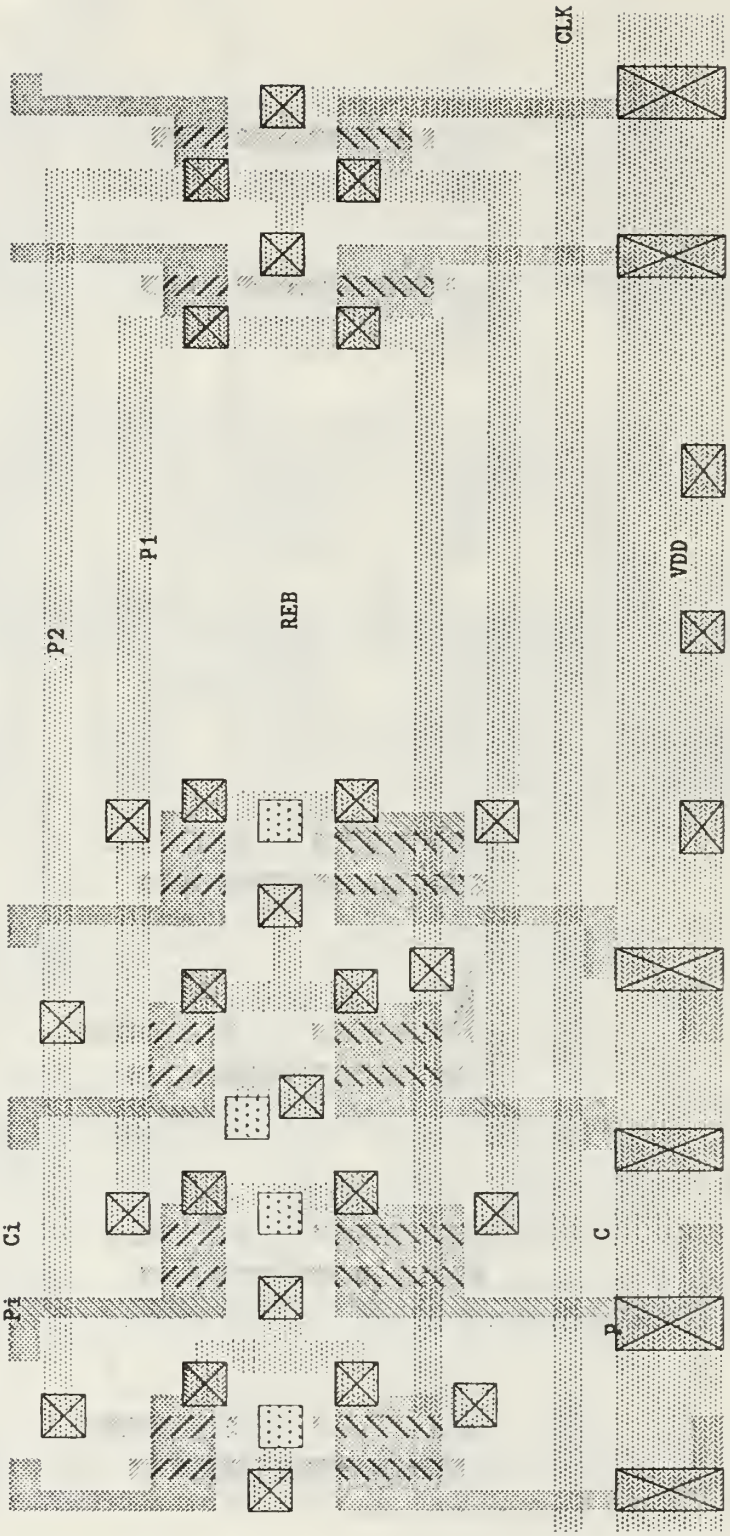


Figure 92. Cell REB layout

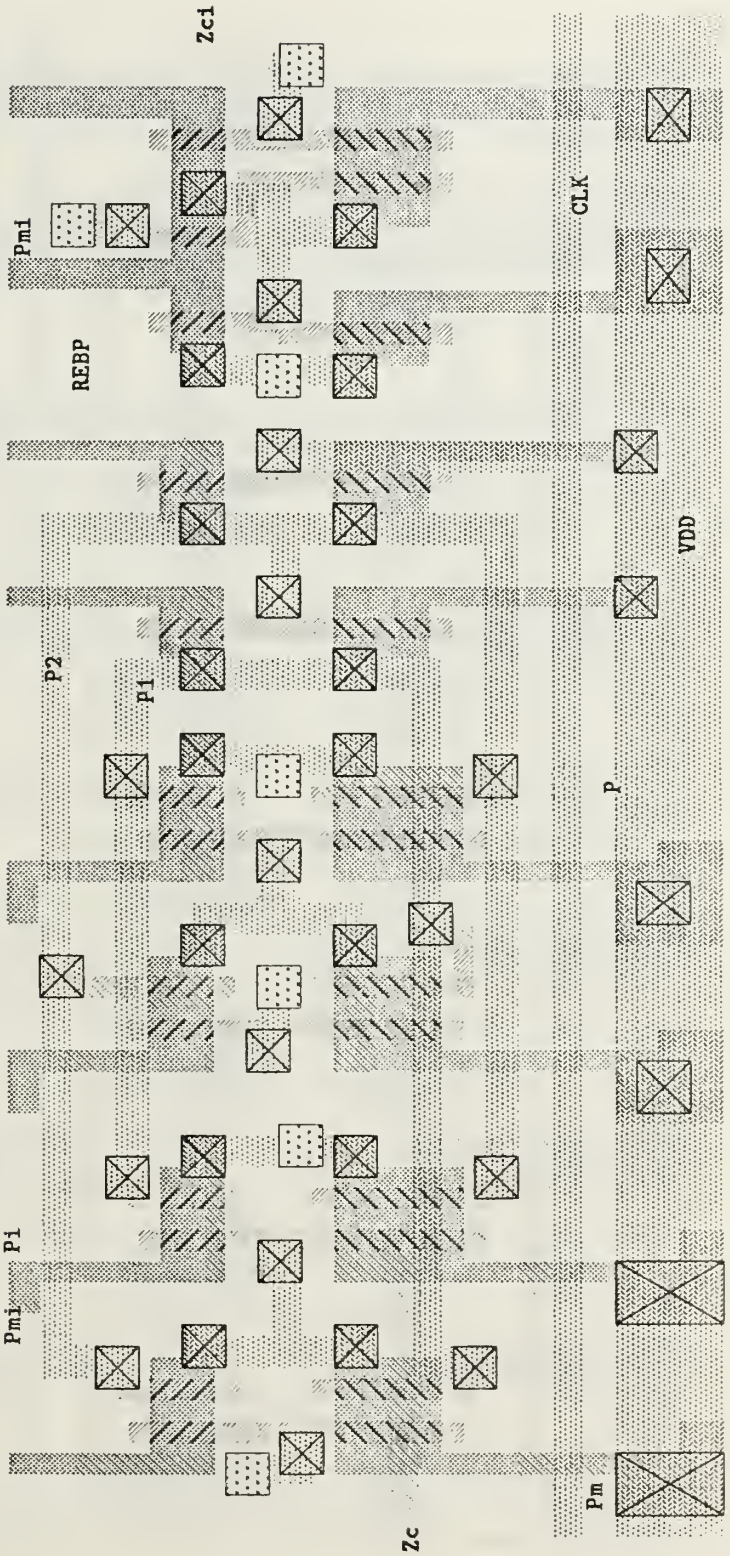


Figure 93. Cell REBP layout

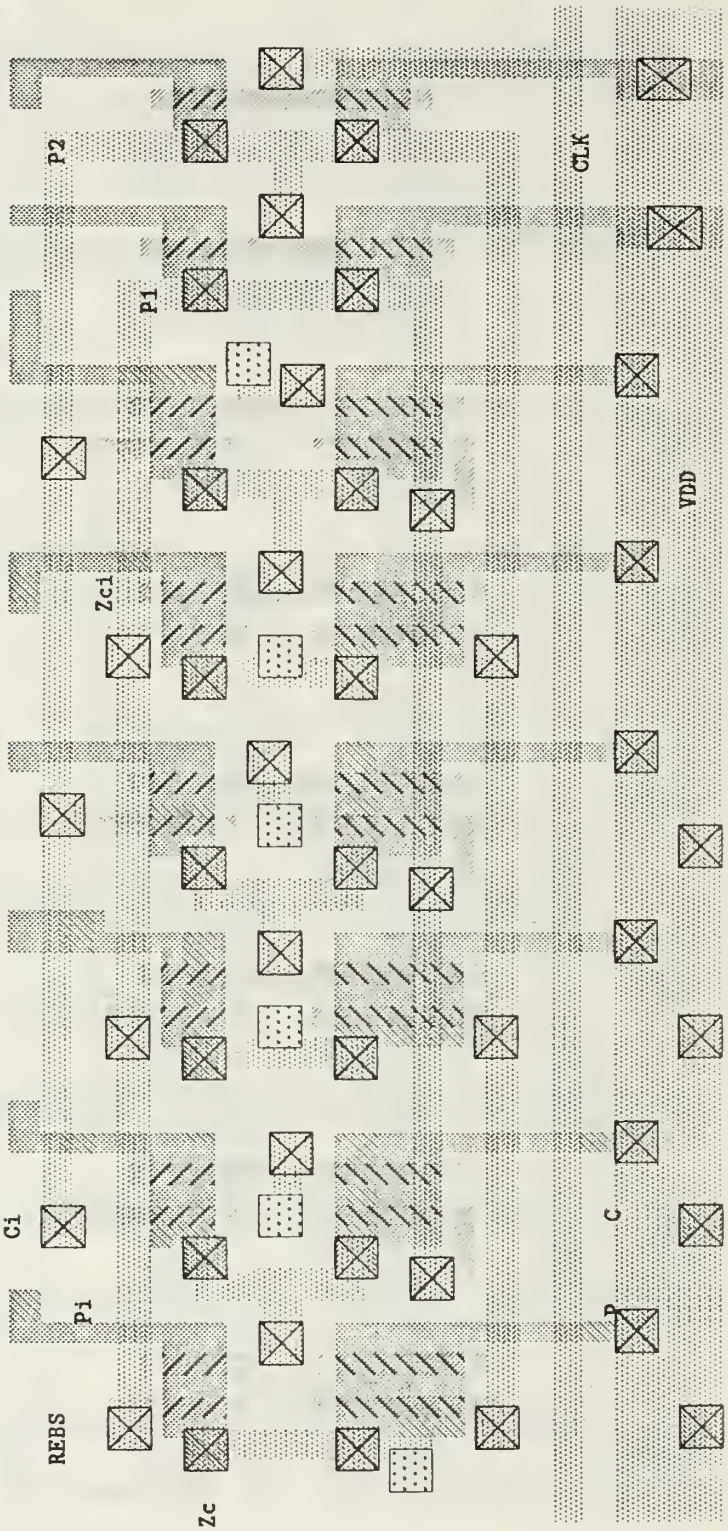


Figure 94. Cell REBS layout

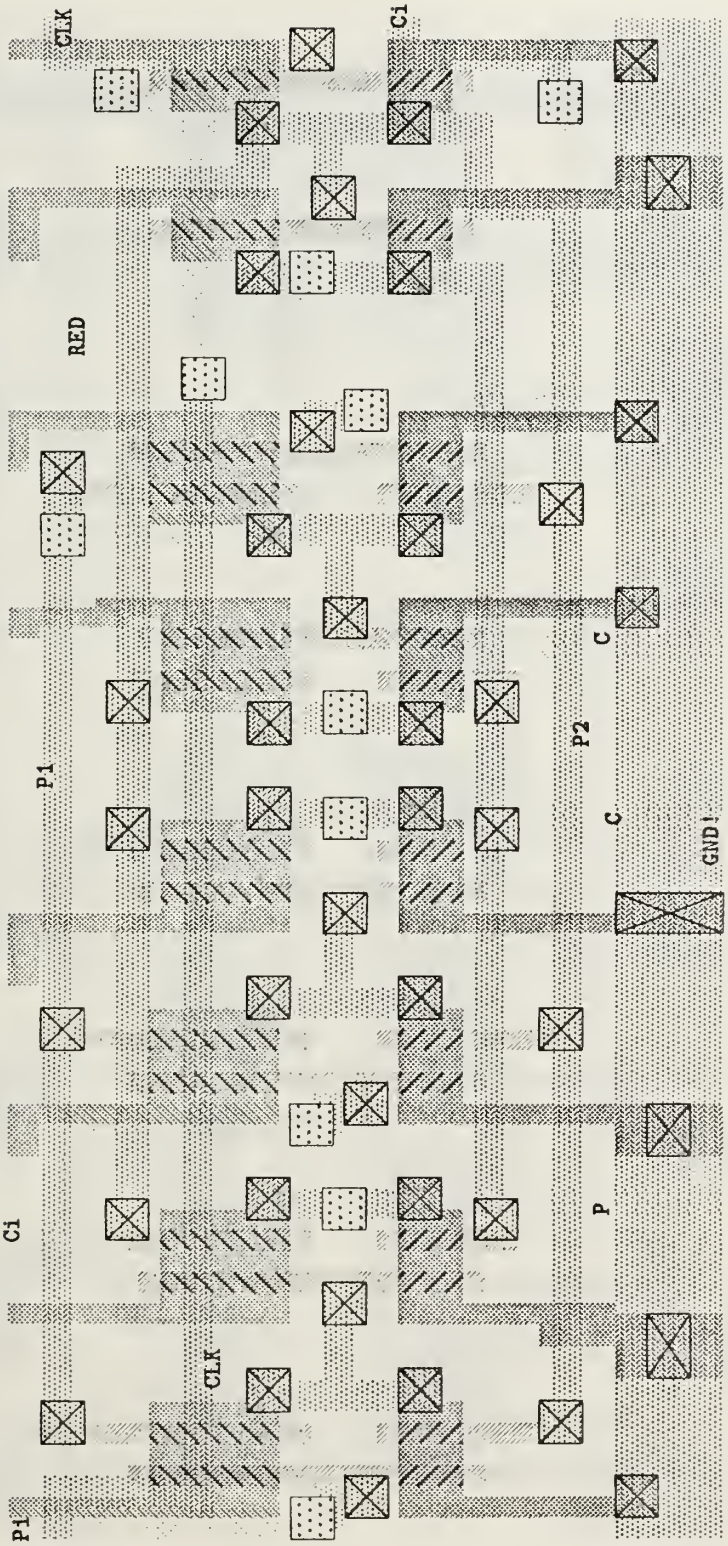


Figure 95. Cell RED layout

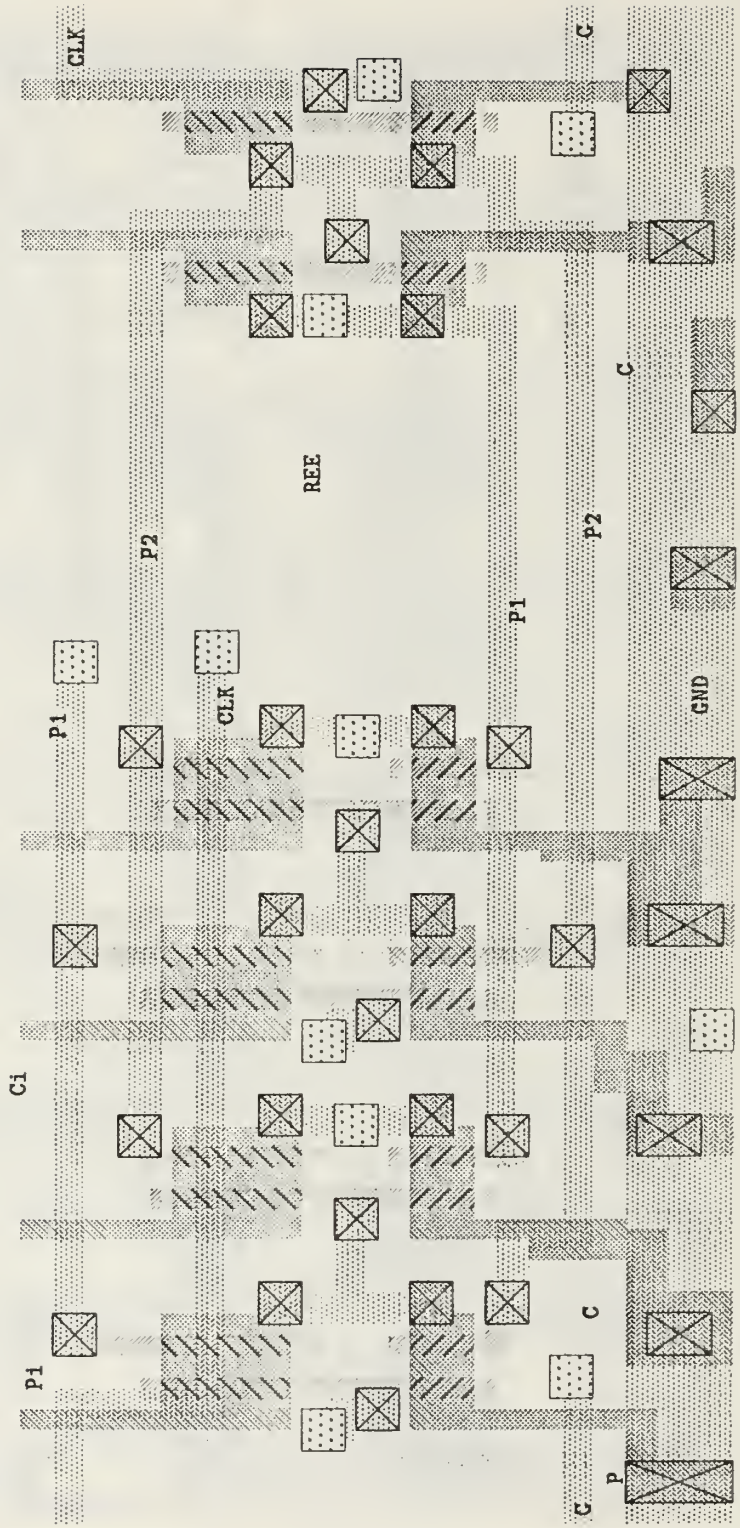


Figure 96. Cell REE layout

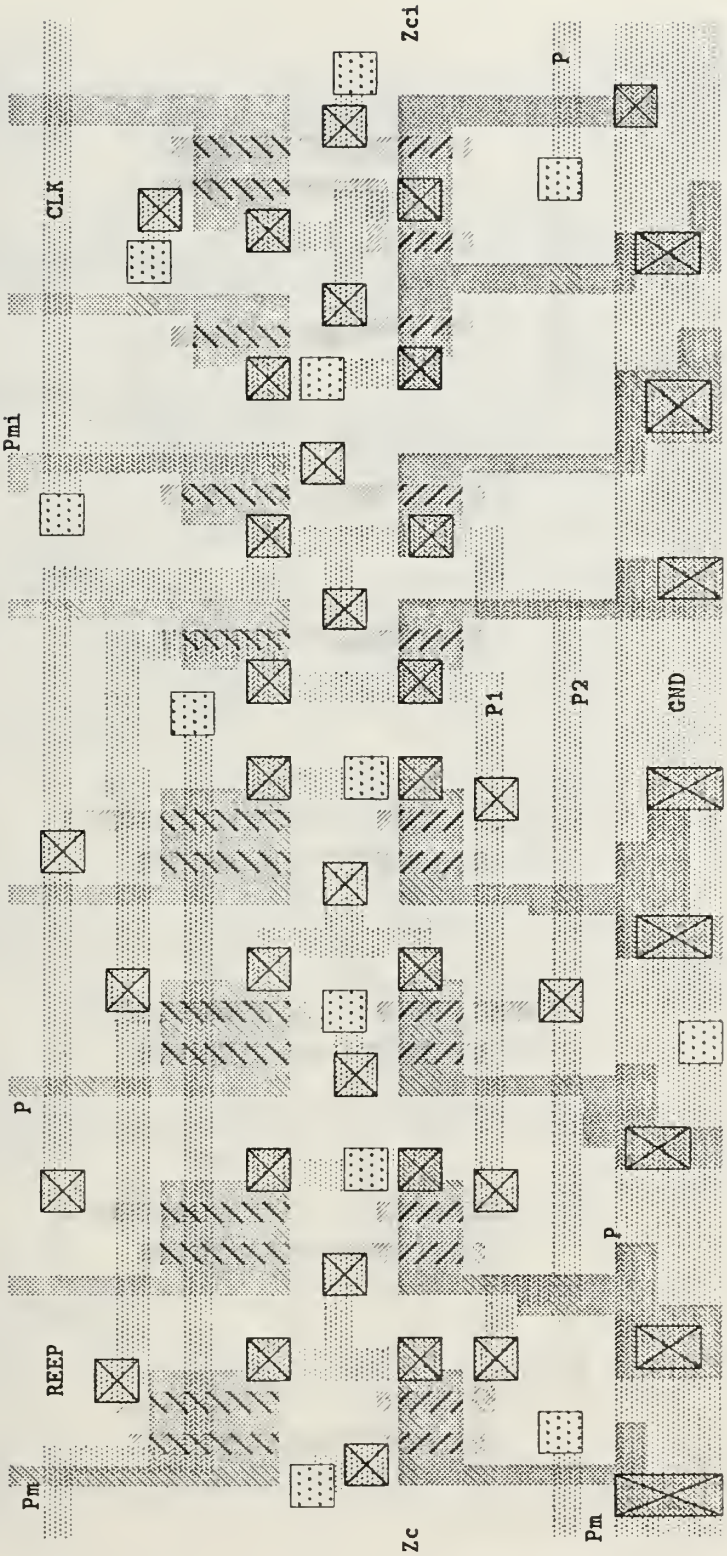


Figure 97. Cell REEP layout

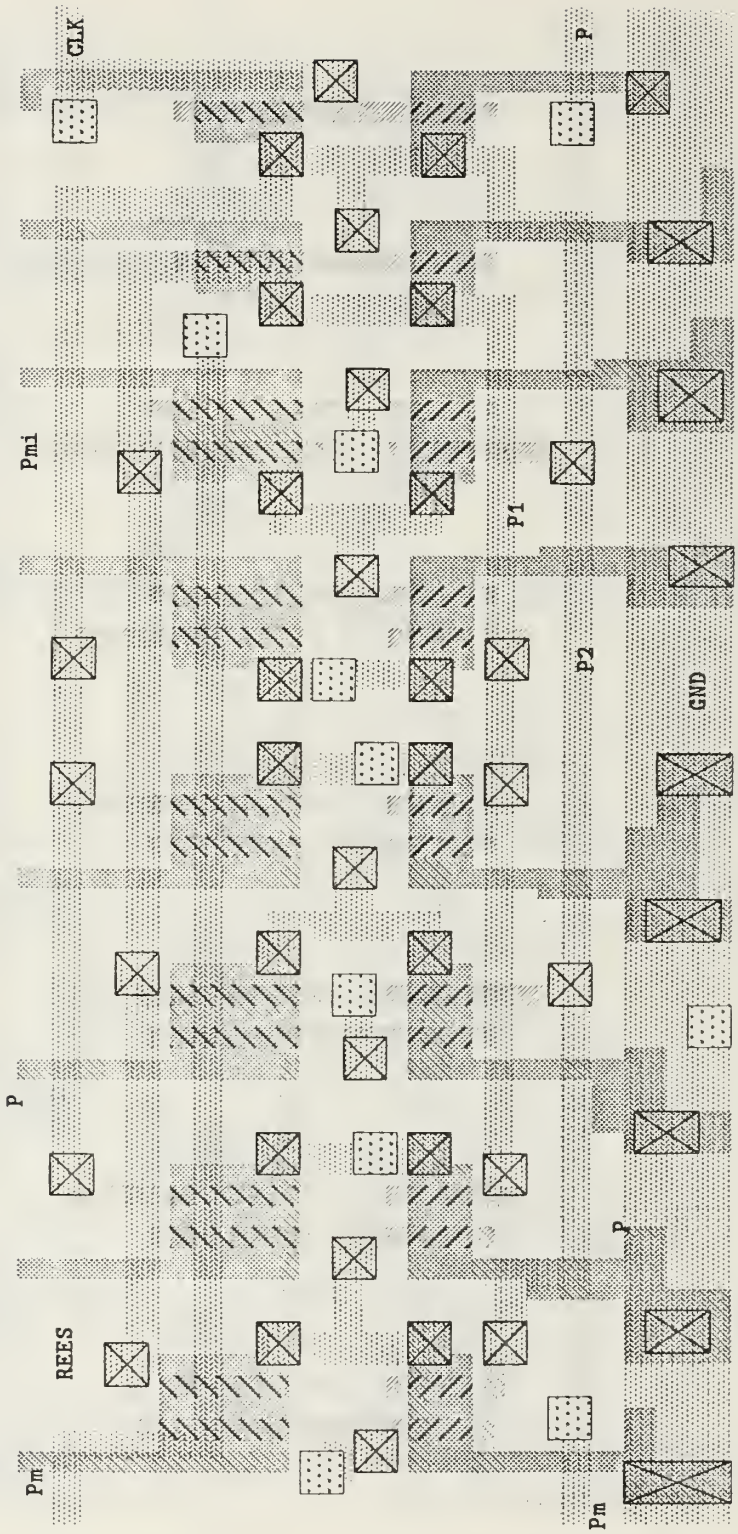


Figure 98. Cell REES layout

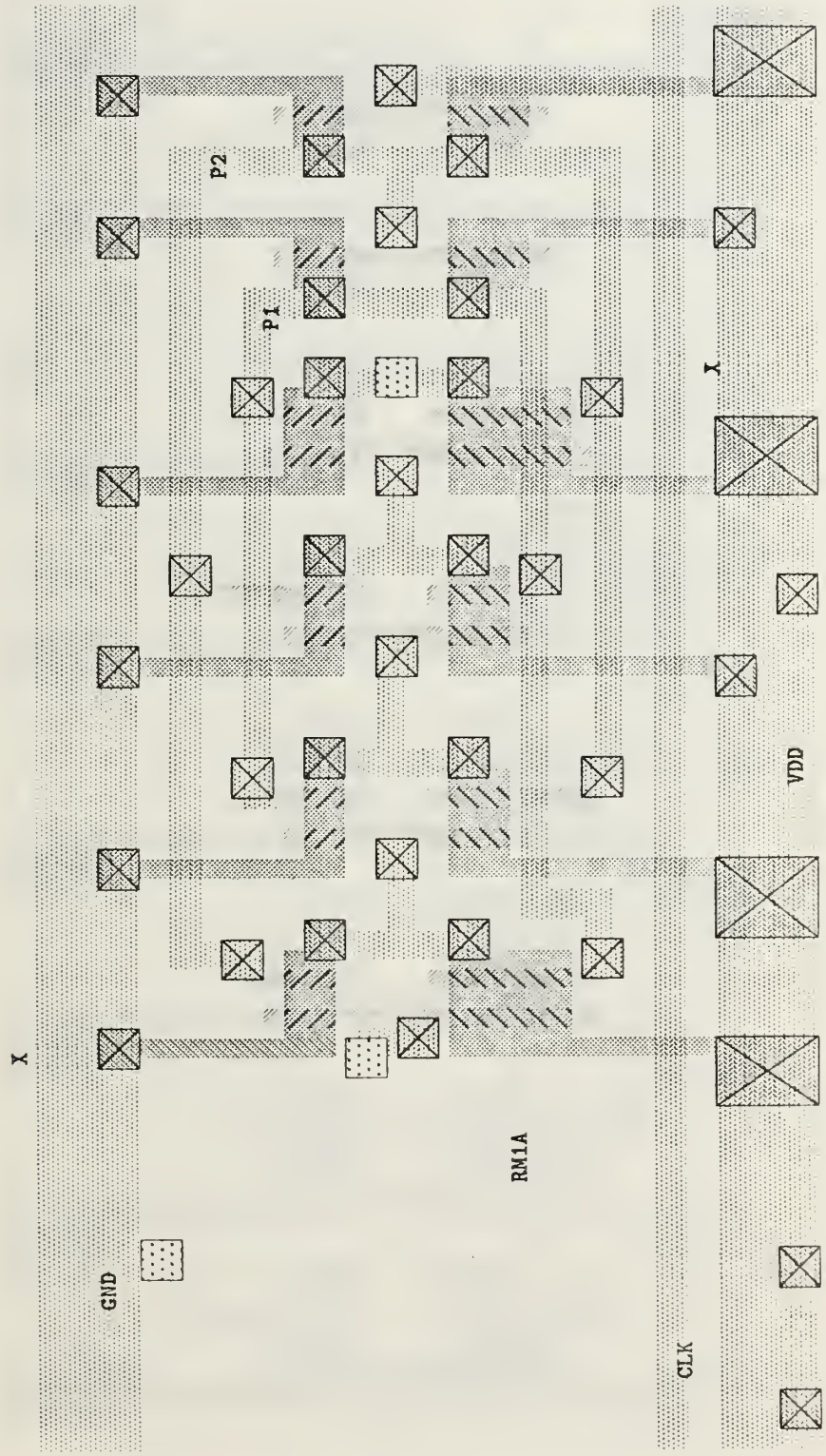


Figure 99. Cell RM1A layout

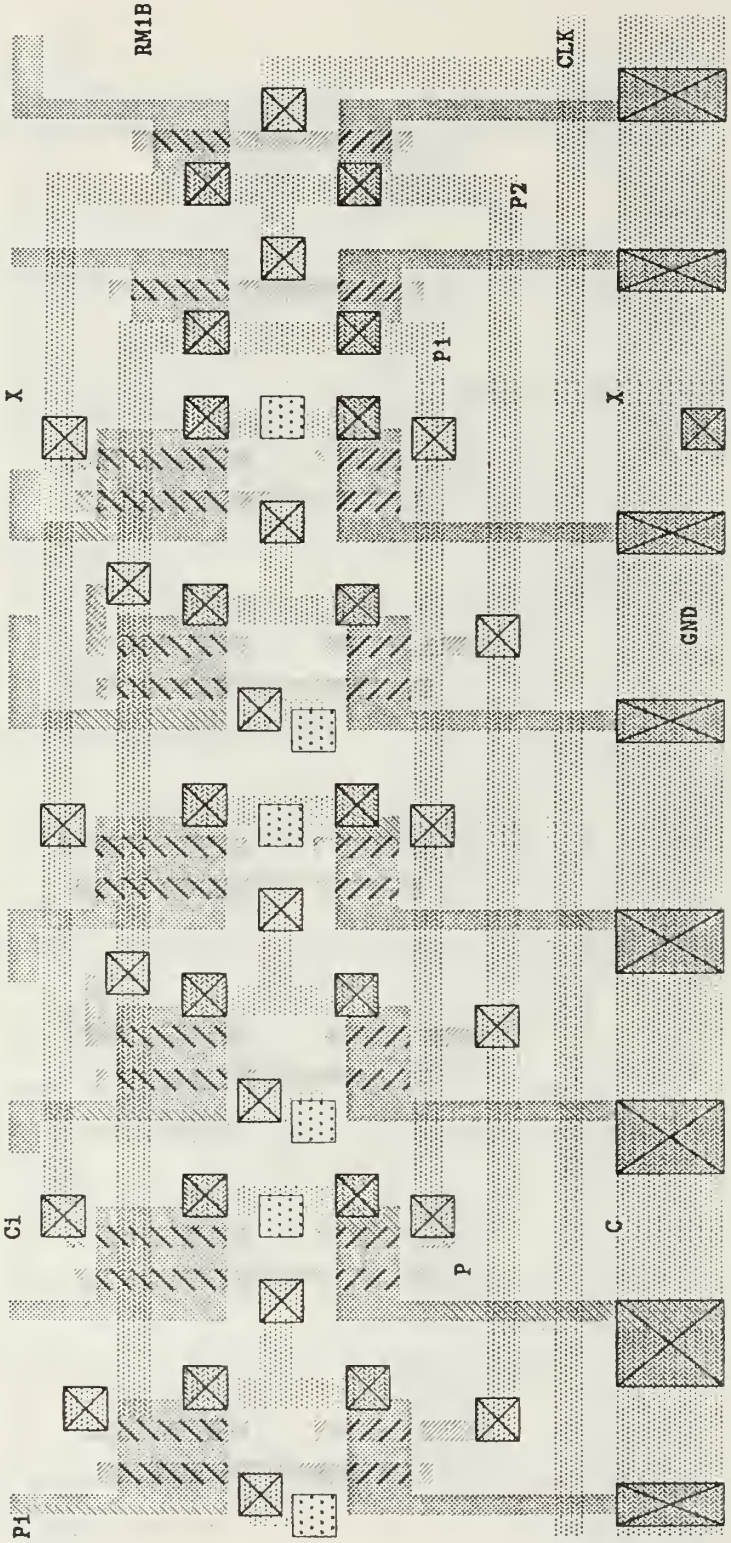


Figure 100. Cell RM1B layout

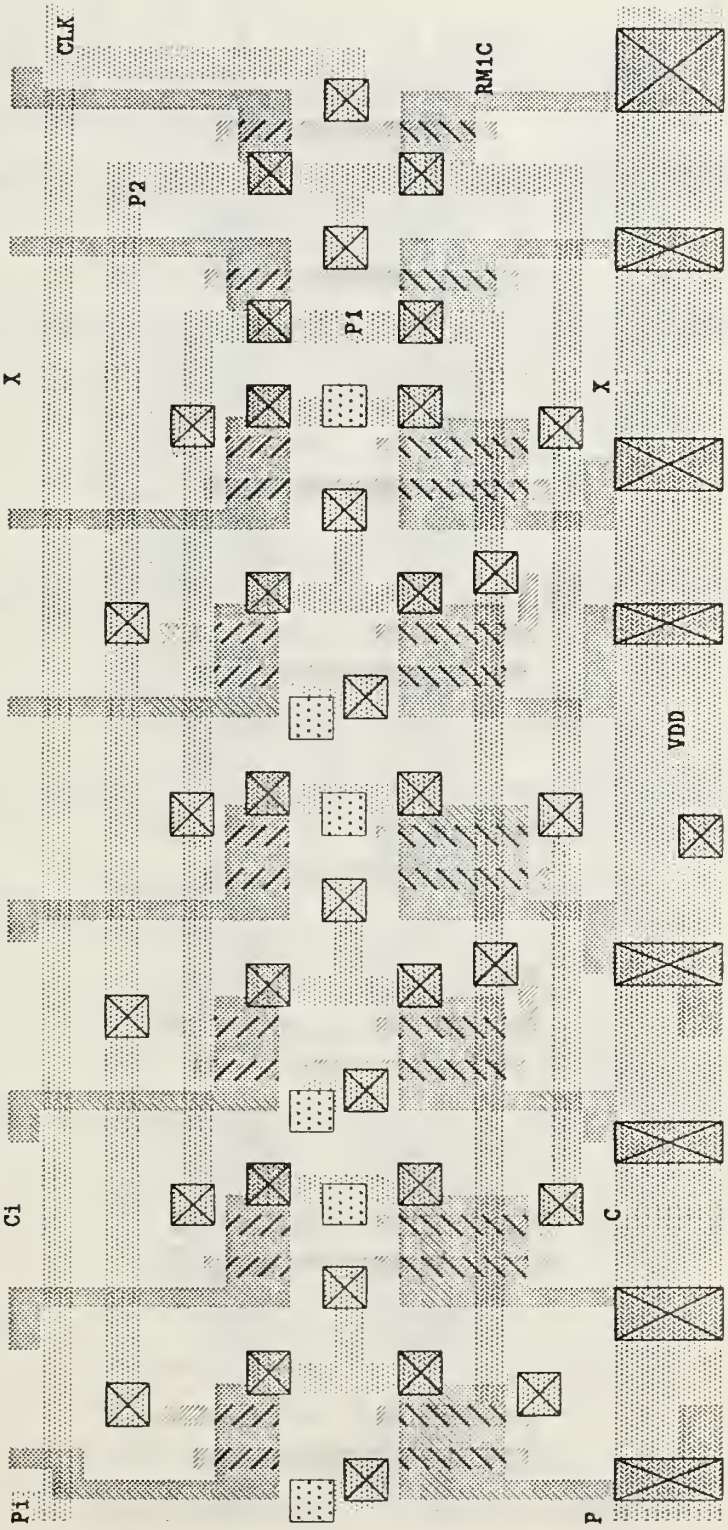


Figure 101. Cell RM1C layout

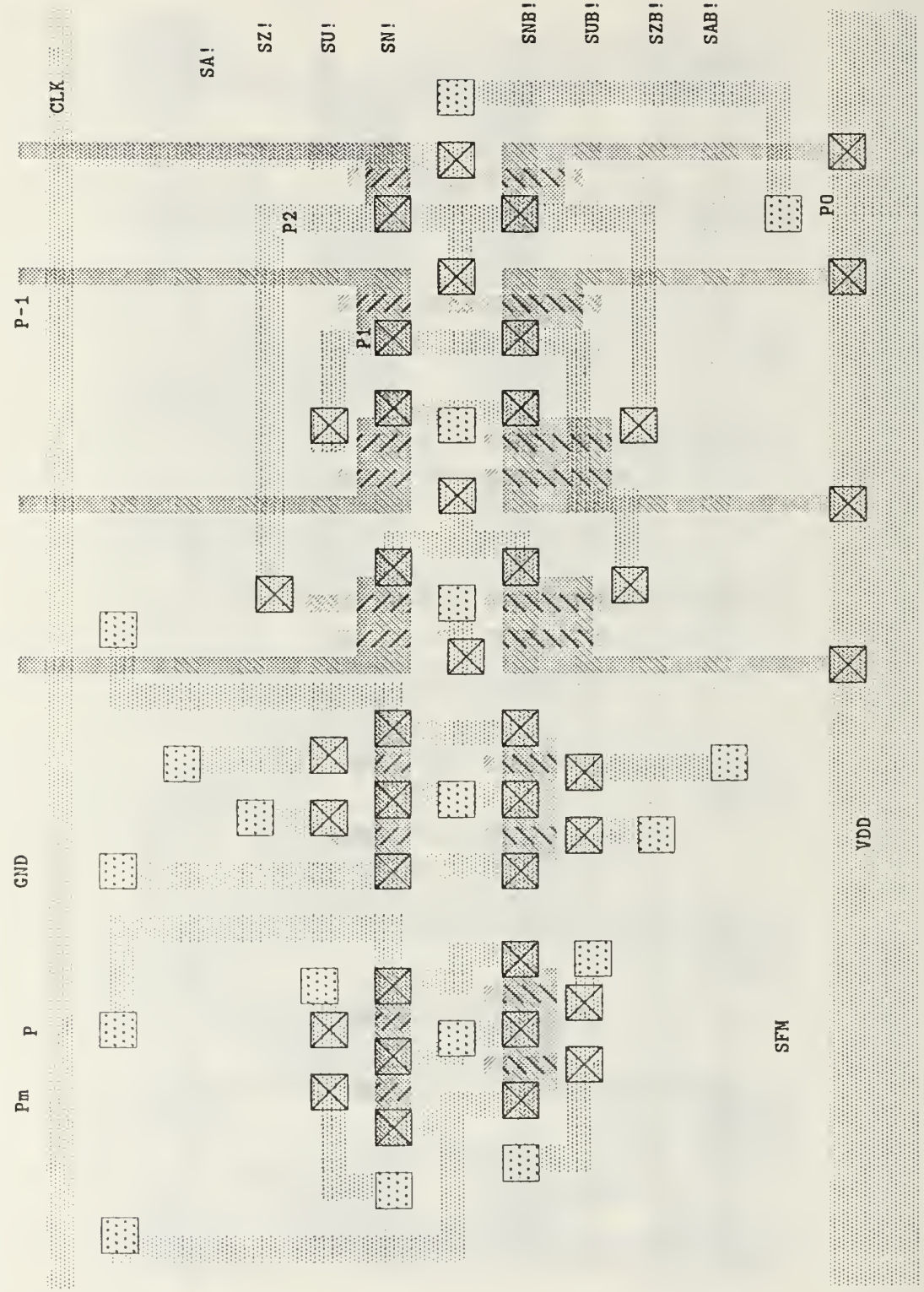


Figure 102. Cell SFM layout

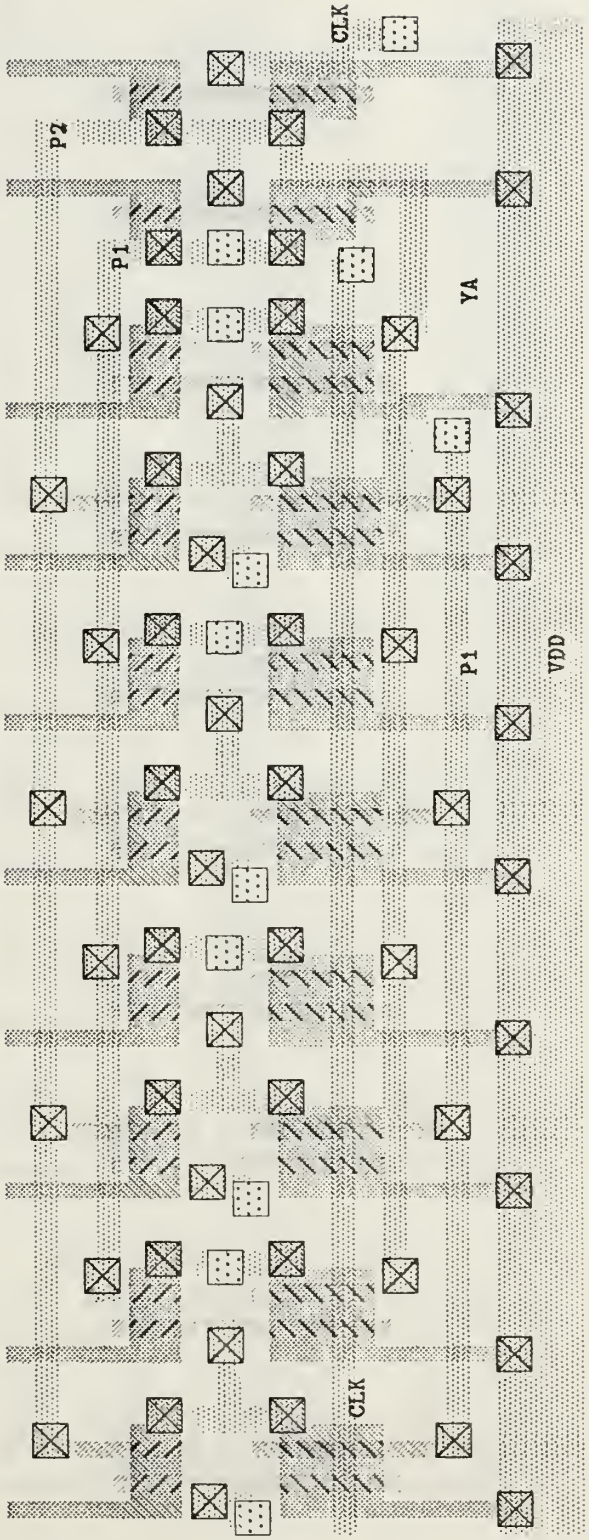


Figure 103. Cell YA layout

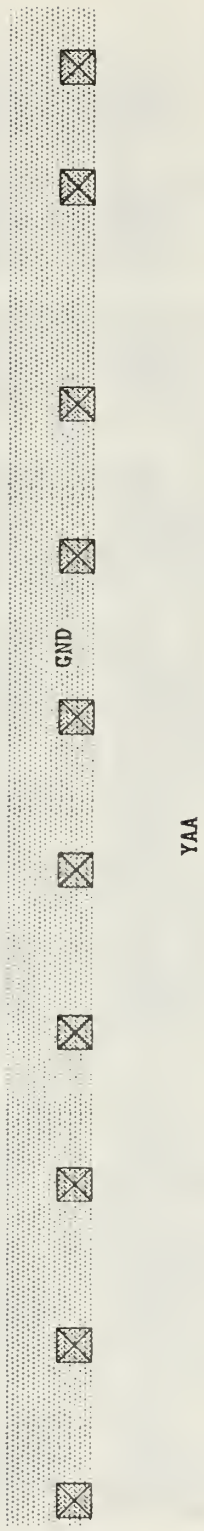


Figure 104. Cell YAA layout

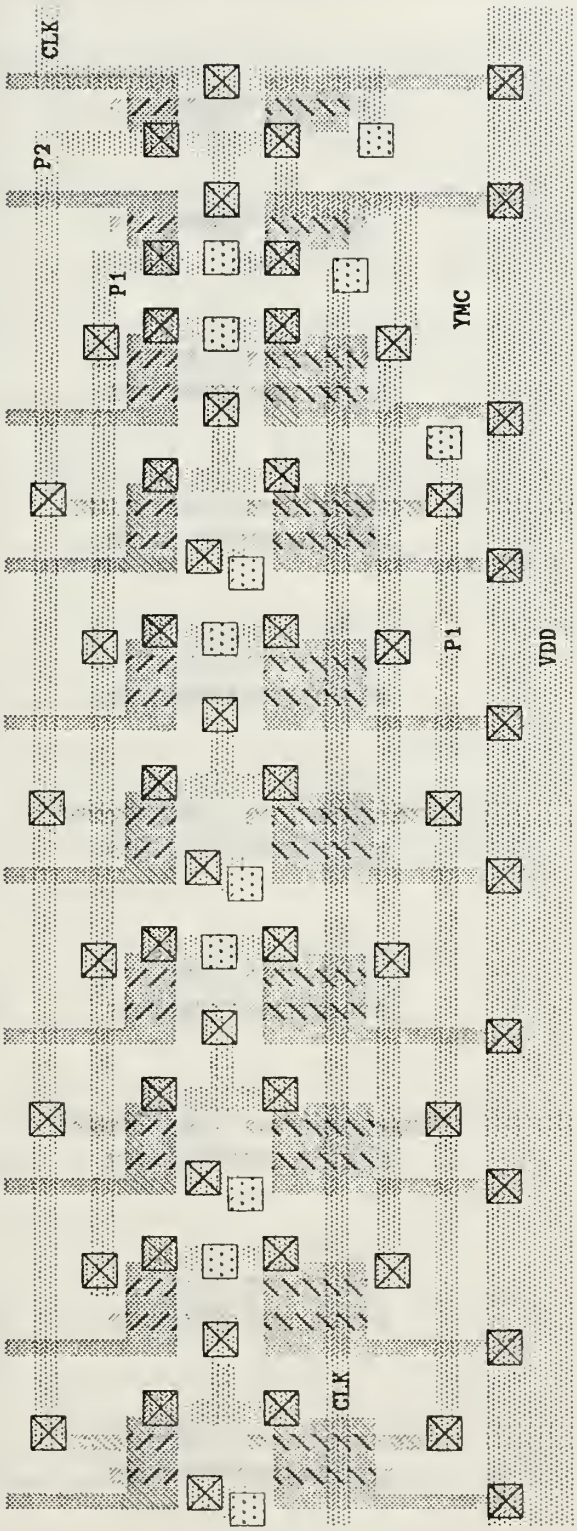


Figure 105. Cell YMC layout

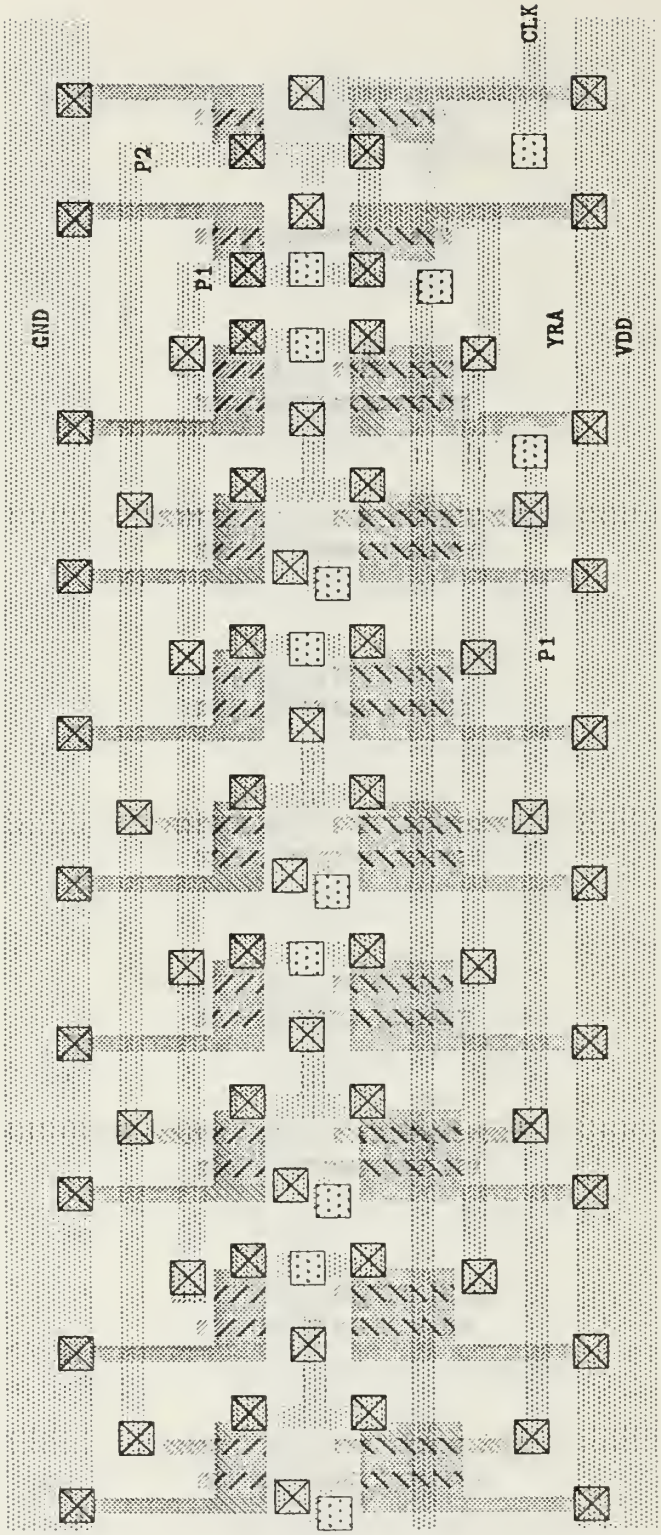


Figure 106. Cell YRA layout

C. EXPONENT SUBTRACTION FUNCTION CELLS.

The standard cells used to perform the exponent subtraction function used in a floating point addition or subtraction exponent compare operation are listed below. An example arrangement of cells to perform a 12-bit exponent subtraction is shown in Figure 37 on page 65. The cells subtract the "B" exponent from the "A" exponent; the difference is labeled D. The two's complement of the difference, labeled Dtc, is also computed. All four values (A, B, D, and Dtc) are stored until the sign of the difference is known. Then the appropriate exponent is selected to be retained, and the five LSBs of D or Dtc are used to generate the control bits for the addition mantissa alignment process. The following list describes the cells and their functions:

Cell CA: See cell description in the exponent addition function area.

Cell CAC: Figure 107 on page 154; a "clock" cell that provides a CLK output to cells in its row by inverting the CLKin signal. The cell has a GND bus running along its lower edge as well as GND and VDD busses running vertically.

Cell CB: See cell description in the exponent addition function area.

Cell CI: See cell description in the exponent addition function area.

Cell CSB: Figure 108 on page 155; a cell used to provide space and connections to GND and VDD busses. There are no active elements in this cell.

Cell ESI: Figure 109 on page 156; a cell composed of three inverter pairs. The inverters drive the control lines (STORA, STORB, and LD1) for the exponent selection and shift control generation cells.

Cell EXSLOG: Figure 110 on page 157; a logic cell that generates the shift network control selection signals. If STORA (store A exponent) is active, the A exponent is selected for storage. If STORB is active, the B exponent is stored. If LD1 (load ones) is active, all alignment control lines are pulled high. Figure 31 on page 55 shows the gate level logic design incorporated into the cell.

Cell RA: See cell description in the exponent addition function area.

Cell RAI: See cell description in the exponent addition function area.

Cell RAIW: See cell description in the exponent addition function area.

Cell RASW: Figure 111 on page 158; a pipeline latch cell that stores an A and a B bit for two clock cycles. The cell also contains the control inverters for the latch.

Cell RAW: See cell description in the exponent addition function area.

Cell RA1: See cell description in the exponent addition function area.

Cell RA2W: Figure 112 on page 159; a pipeline latch cell that stores two bits for a single clock cycle.

Cell RA3: Figure 113 on page 160; a pipeline latch cell used to store the LD1, STORA, and STORB control signals prior to use.

Cell RA4: Figure 114 on page 161; a pipeline latch cell used to store four values (A, B, D, and Dtc).

Cell RA4CL: Figure 115 on page 162; a pipeline latch cell used to store four values. It is designed to be used to store the D and Ci values of the last logic cell in a pipeline stage. The D and Ci are used in the next pipeline stage to compute the two's complement of D.

Cell RA4W: Figure 116 on page 163; a pipeline latch cell that stores the D, D1s, Dtc1s, and Ci signals used in computing the two's complement of the difference, and the ones sensing scheme.

Cell RB: See cell description in the exponent addition function area.

Cell RBW: See cell description in the exponent addition function area.

Cell RB1: See cell description in the exponent addition function area.

Cell RB4: Figure 117 on page 164; a cell similar in function to cell RA4, but with a VDD bus running along its lower edge.

Cell RI4: Figure 118 on page 165; a pipeline latch cell that stores two values.

Cell RI4W: Figure 119 on page 166; a cell similar to cell RI4, but slightly "wider".

Cell SAE: Figure 120 on page 167; a subtraction cell that produces the difference, D, by subtracting B from A.

Cell SAEL: Figure 121 on page 168; a subtraction cell designed to be the last cell on the subtraction process. (The CARRY-OUT signal is not necessary.)

Cell SAER: Figure 122 on page 169; a subtraction cell designed to be the first logic cell in a pipeline stage.

Cell SAL4: Figure 123 on page 170; a subtraction cell used as the last logic cell in a pipeline stage.

Cell SAR4: Figure 124 on page 171; a subtraction cell used as the first logic cell in the subtraction process. The Ci (CARRY-IN) line is pulled high to provide the increment required in two's complement generation.

Cell SA4: Figure 125 on page 172; a subtraction cell.

Cell SBCL: Figure 126 on page 173; a subtraction cell with functions similar to cell SAL4, but with a VDD bus running along its lower edge.

Cell SBE: Figure 127 on page 174; a subtraction cell.

Cell SBEL: Figure 128 on page 175; a subtraction cell with functions similar to cell SAL4, but with a VDD bus running along its lower edge.

Cell SCA: Figure 129 on page 176; a logic cell that computes the two's complement of the difference, Dtc, and also performs the ones sensing function of Figure 30 on page 54 for both the difference and the two's complement of the difference.

Cell SCAR: Figure 130 on page 177; a logic cell that generates Dtc and also serves as the first logic cell in the ones sensing function.

Cell SCB: Figure 131 on page 178; a cell similar in function to cell SCA, but with a VDD bus running along its lower edge.

Cell SCBL: Figure 132 on page 179; similar in function to cell SCB, but designed to be the first logic cell in a pipeline stage.

Cell SCBW: Figure 133 on page 180; a logic cell similar to cell SCB.

Cell STC: Figure 134 on page 181; a logic cell that generates Dtc.

Cell STCL: Figure 135 on page 182; a logic cell similar to cell STC, but designed to be the first logic cell in a pipeline stage.

Cell STCR: Figure 136 on page 183; a cell that generates Dtc. Designed to generate the two's complement of the least significant difference bit, D0. (Dtc0 = D0; no logic is necessary.)

Cell SVB: Figure 137 on page 184; a selection cell composed of two transmission gates. The control lines select whether the A exponent or the B exponent is stored.

Cell SVBW: Figure 138 on page 185; similar in function to SVB, but slightly wider.

Cell SVBWW: Figure 139 on page 186; similar to cell SVBW, but slightly wider.

Cell SV4B: Figure 140 on page 187; a selection cell that carries out the exponent selection and mantissa alignment control signal generation processes. Figure 31 on page 55 shows the gate level design incorporated into the cell.

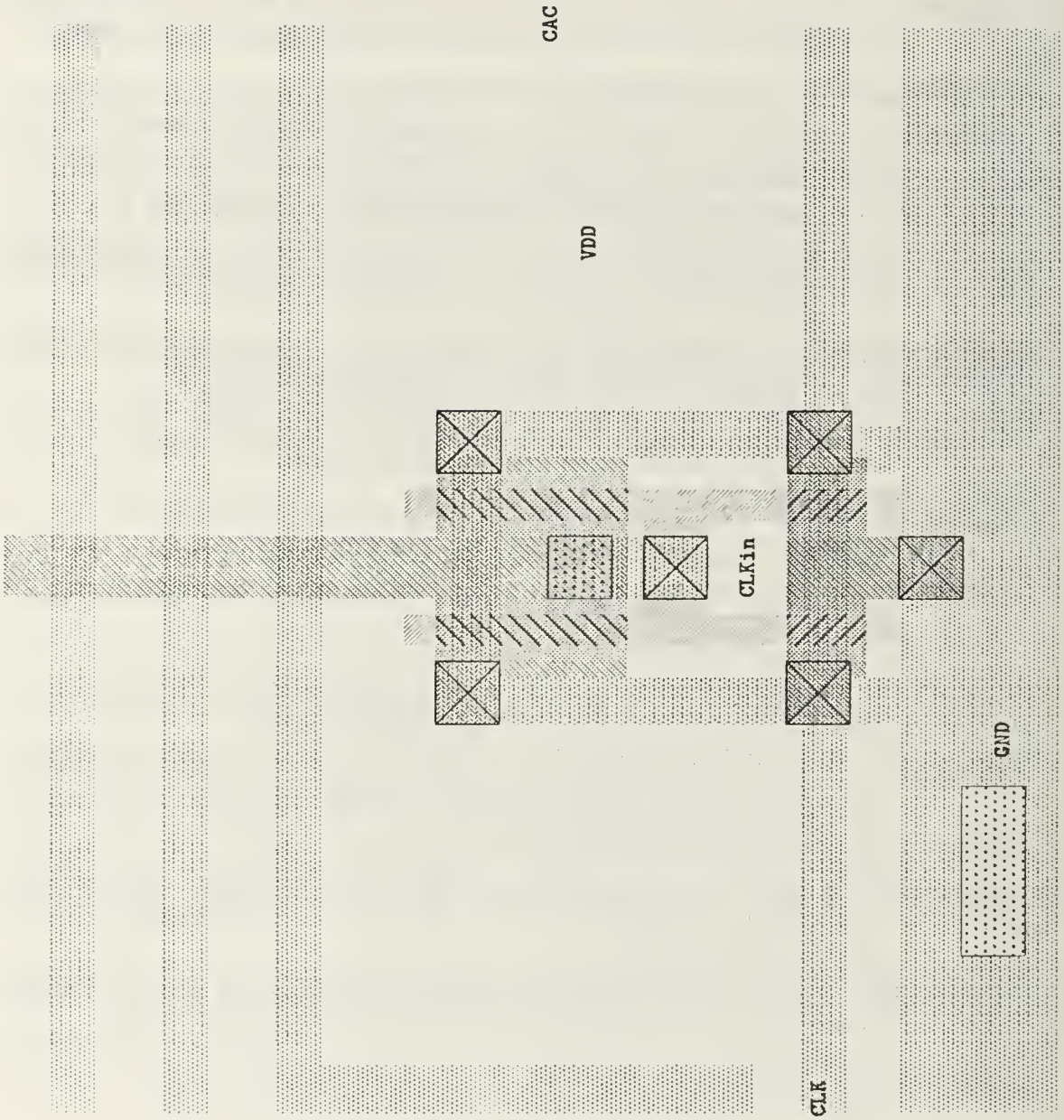


Figure 107. Cell CAC layout

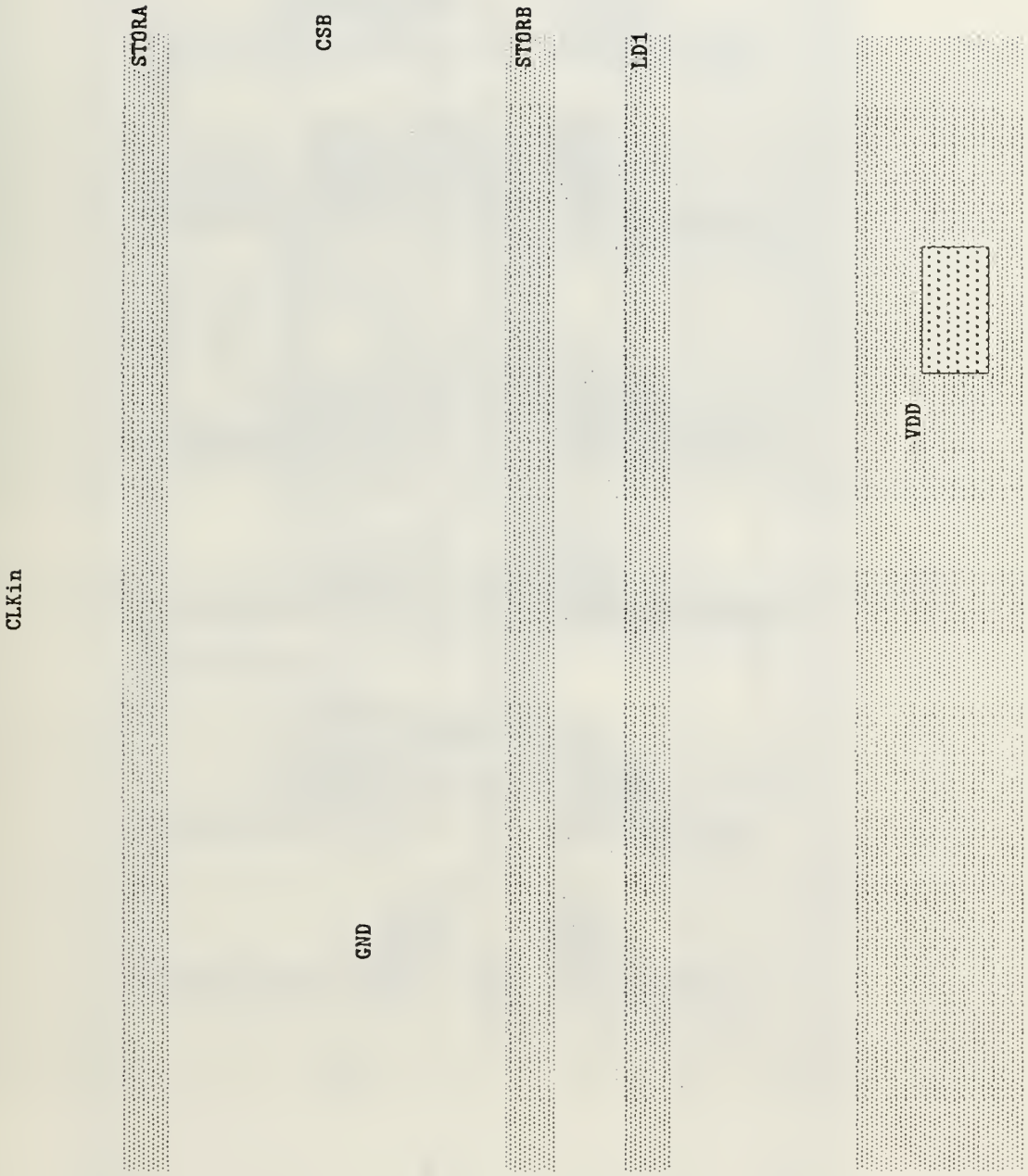


Figure 108. Cell CSB layout

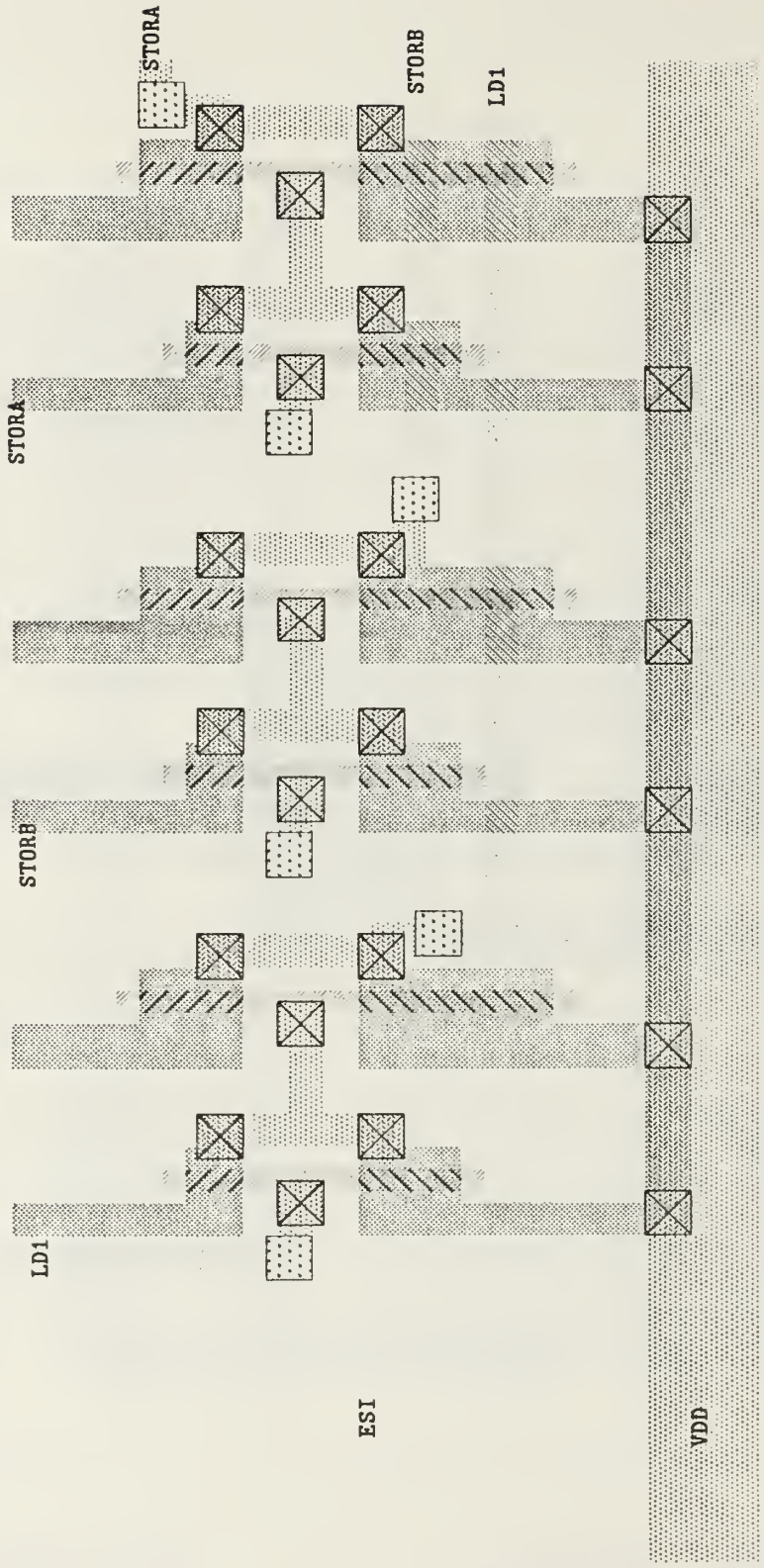


Figure 109. Cell ESI layout

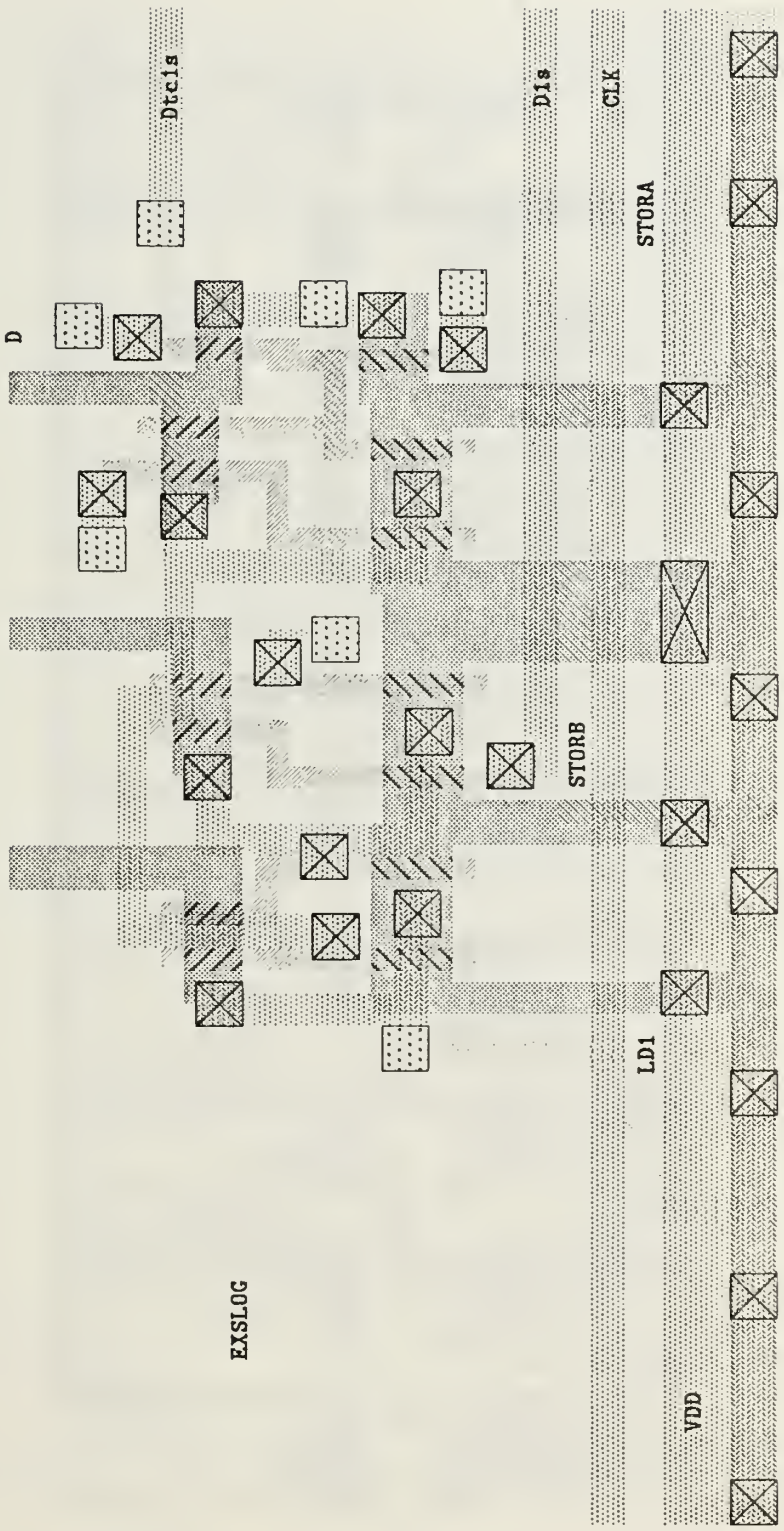


Figure 110. Cell EXSLOG layout

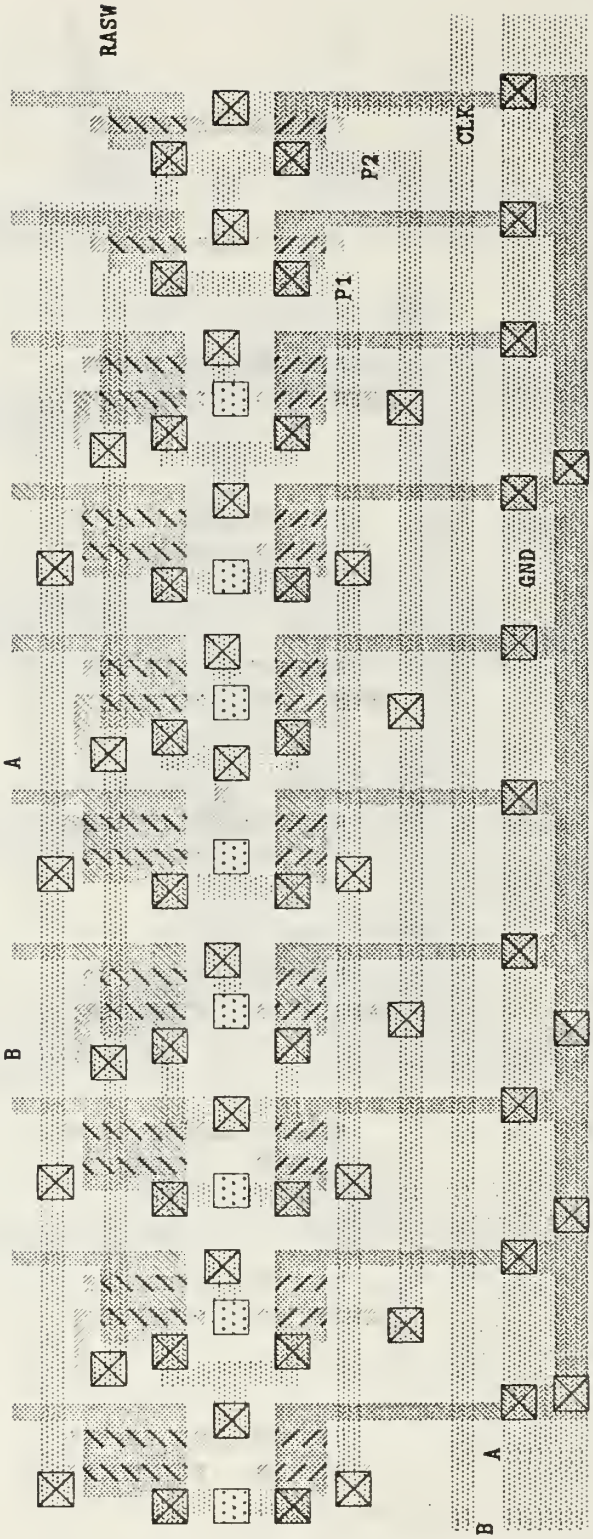


Figure 111. Cell RASW layout

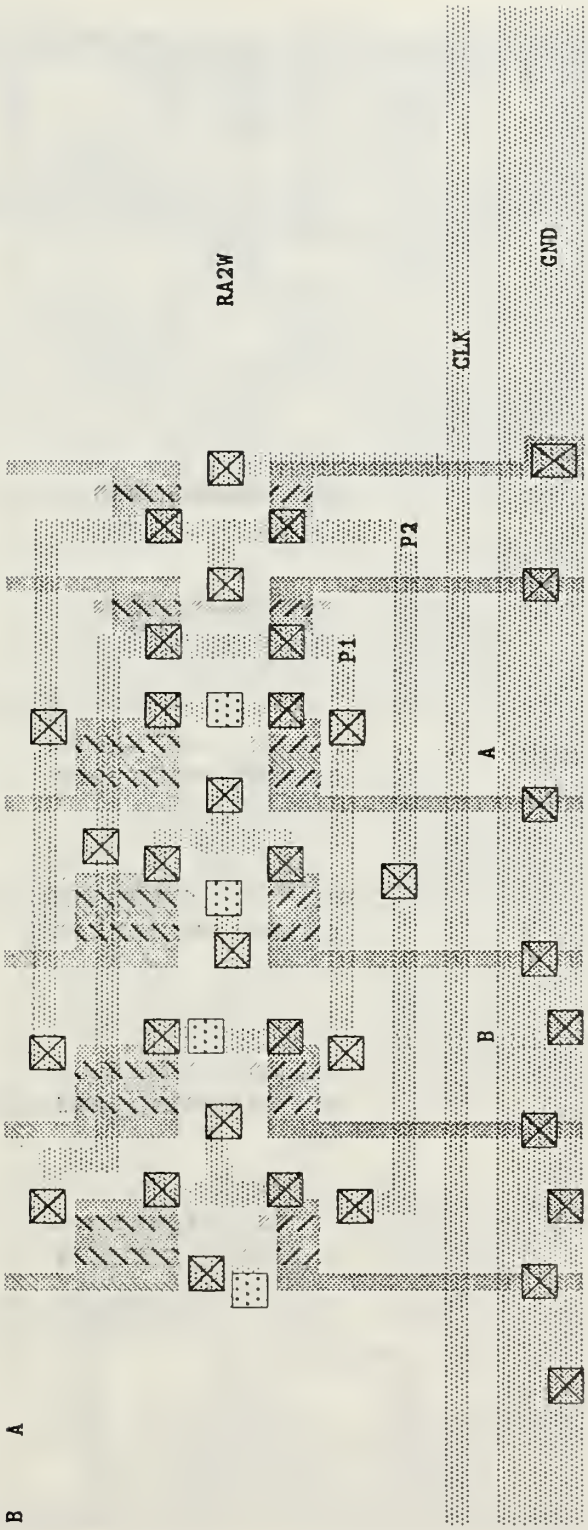


Figure 112. Cell RA2W layout

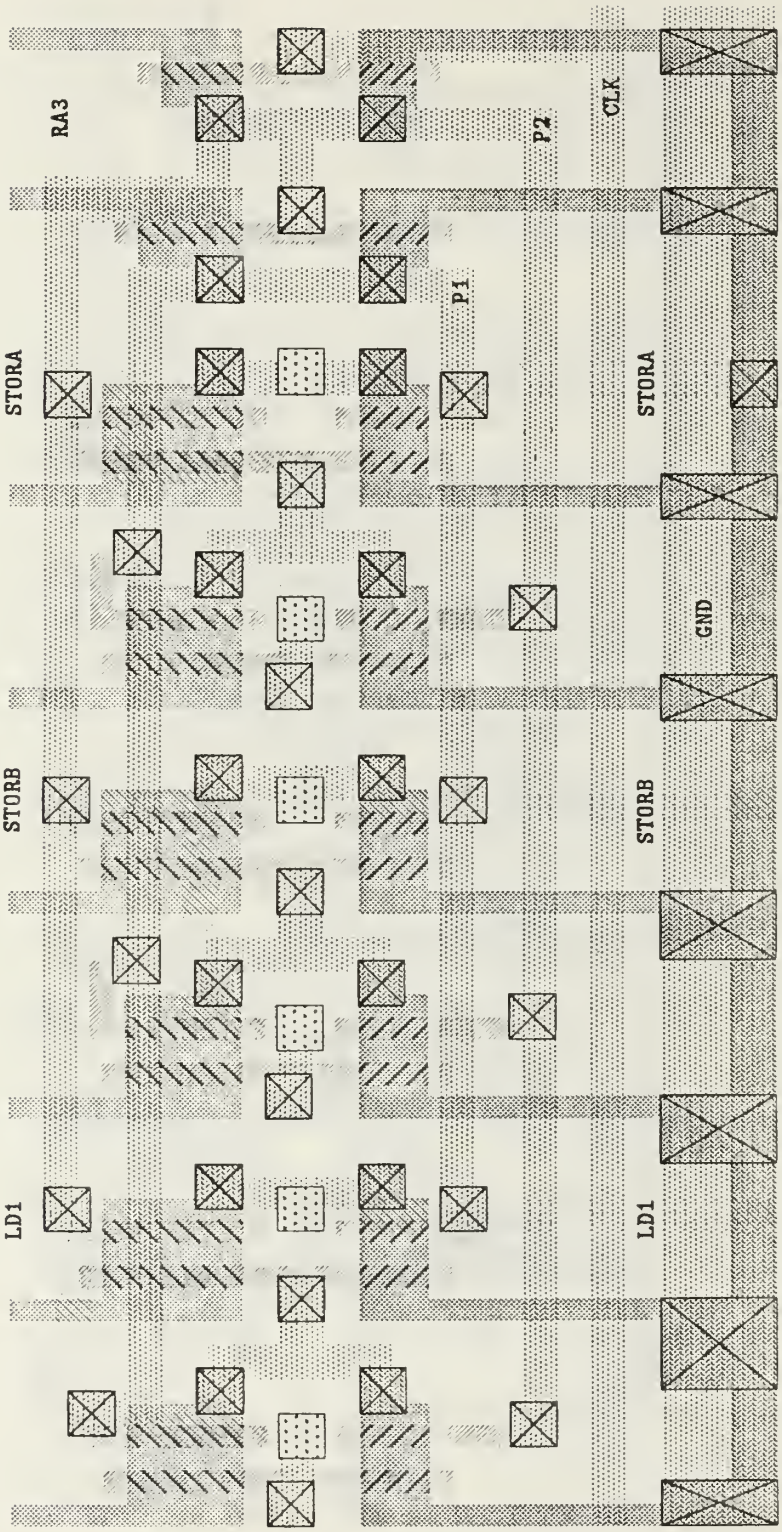


Figure 113. Cell RA3 layout

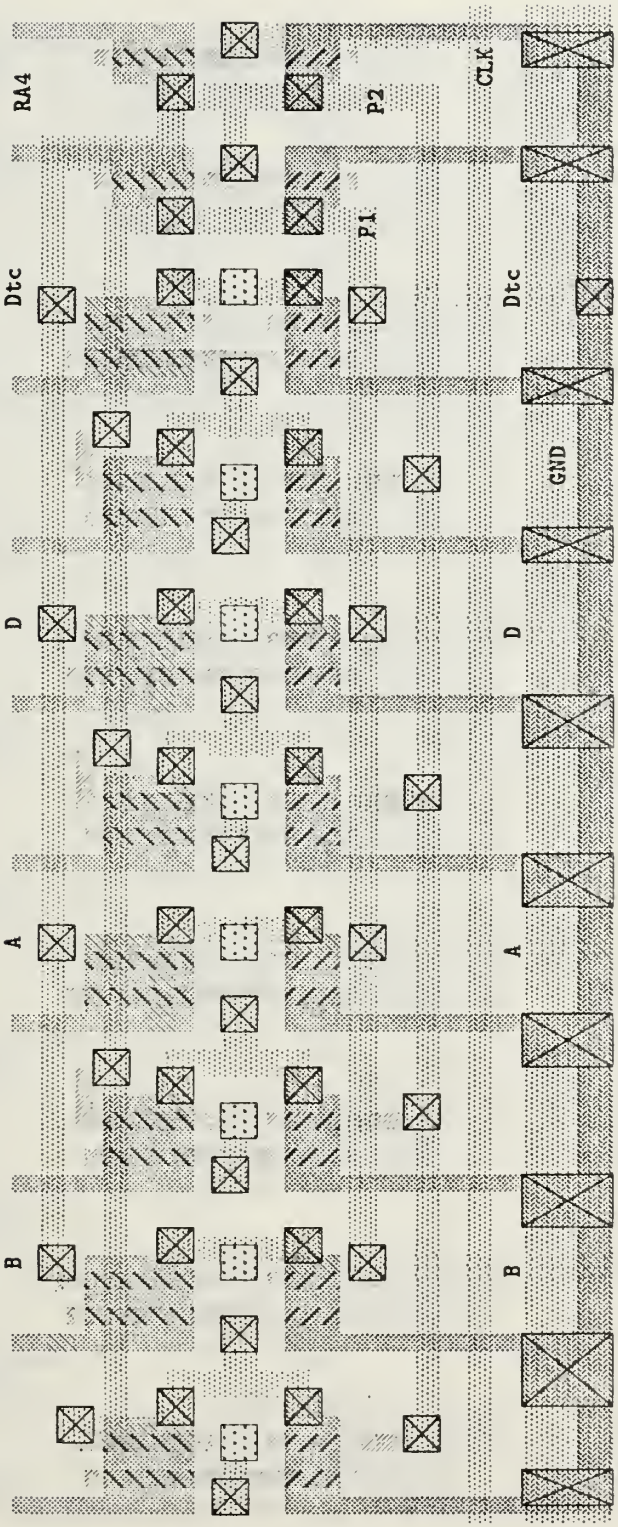


Figure 114. Cell RA4 layout

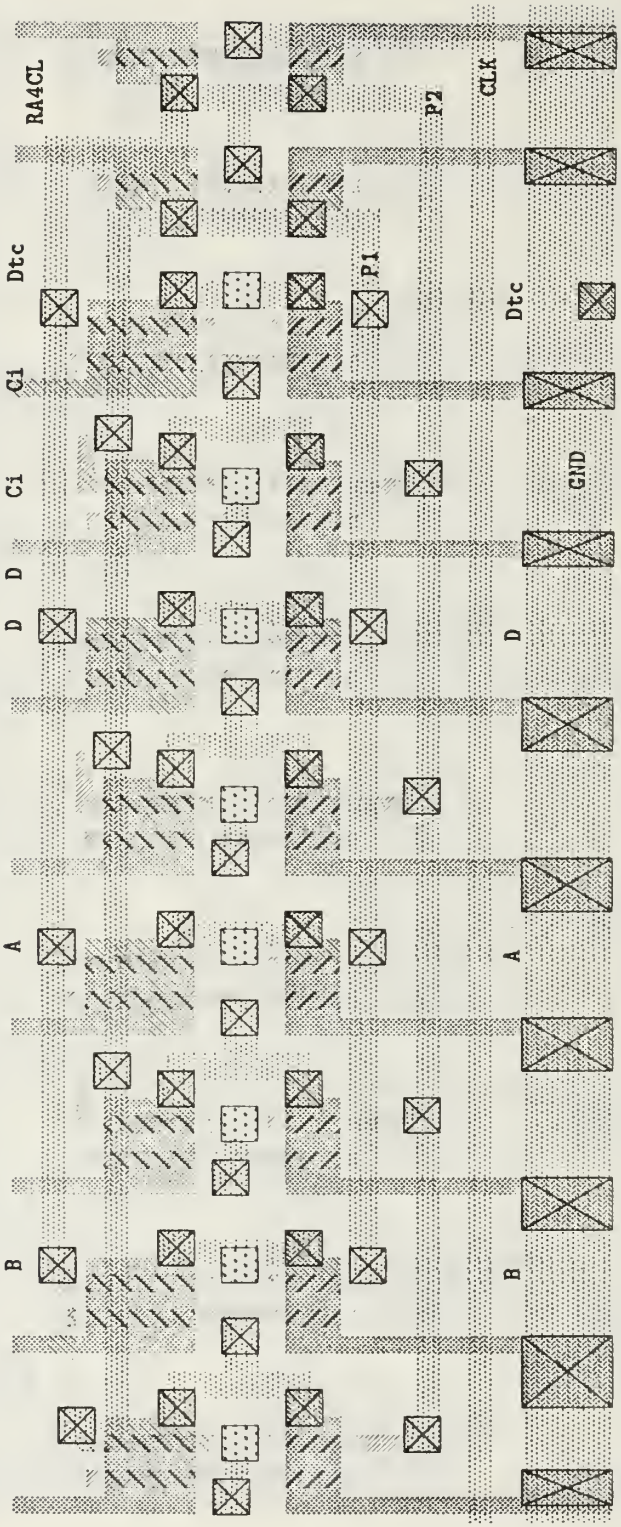


Figure 115. Cell RA4CL layout

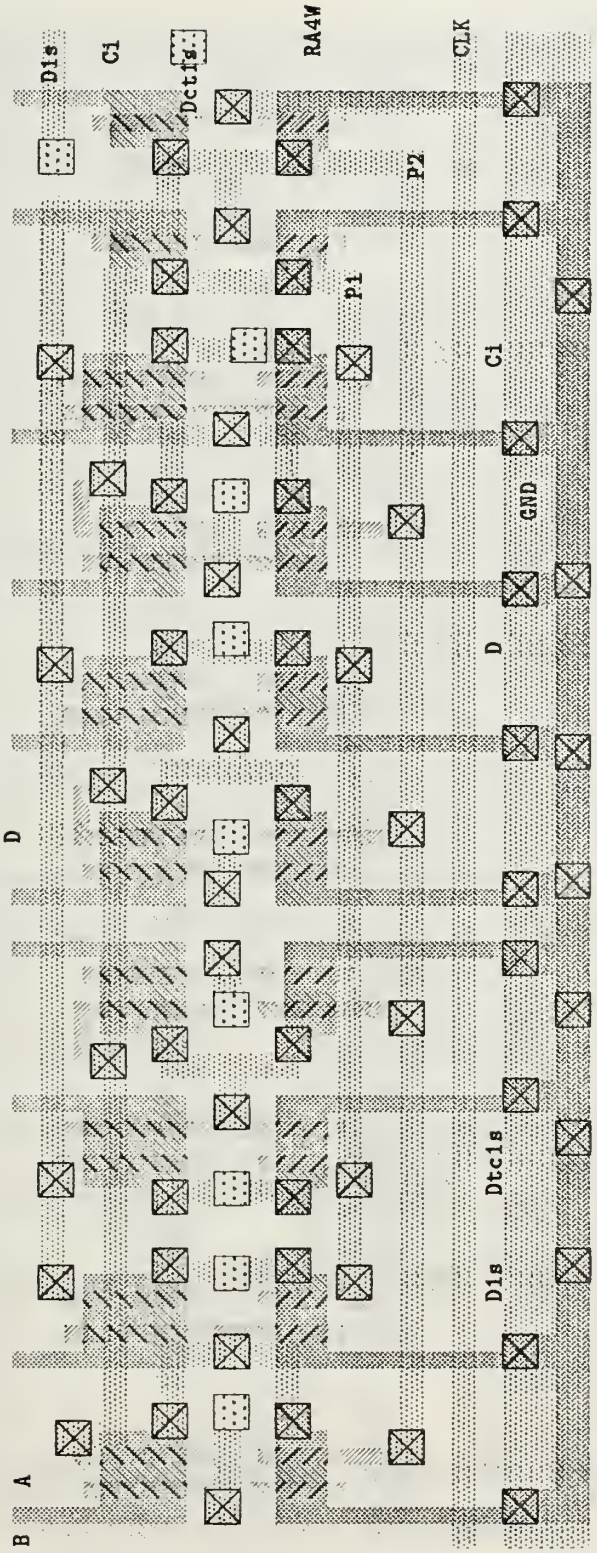


Figure 116. Cell RA4W layout

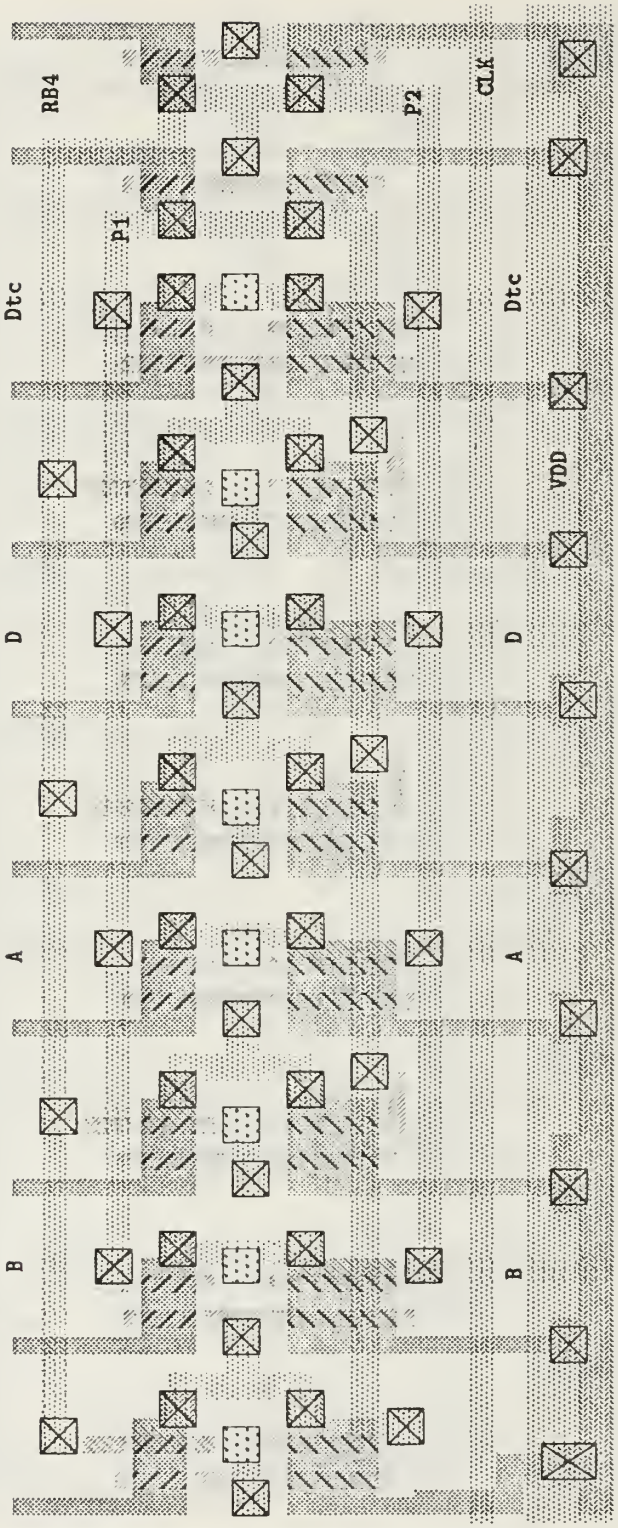


Figure 117. Cell RB4 layout

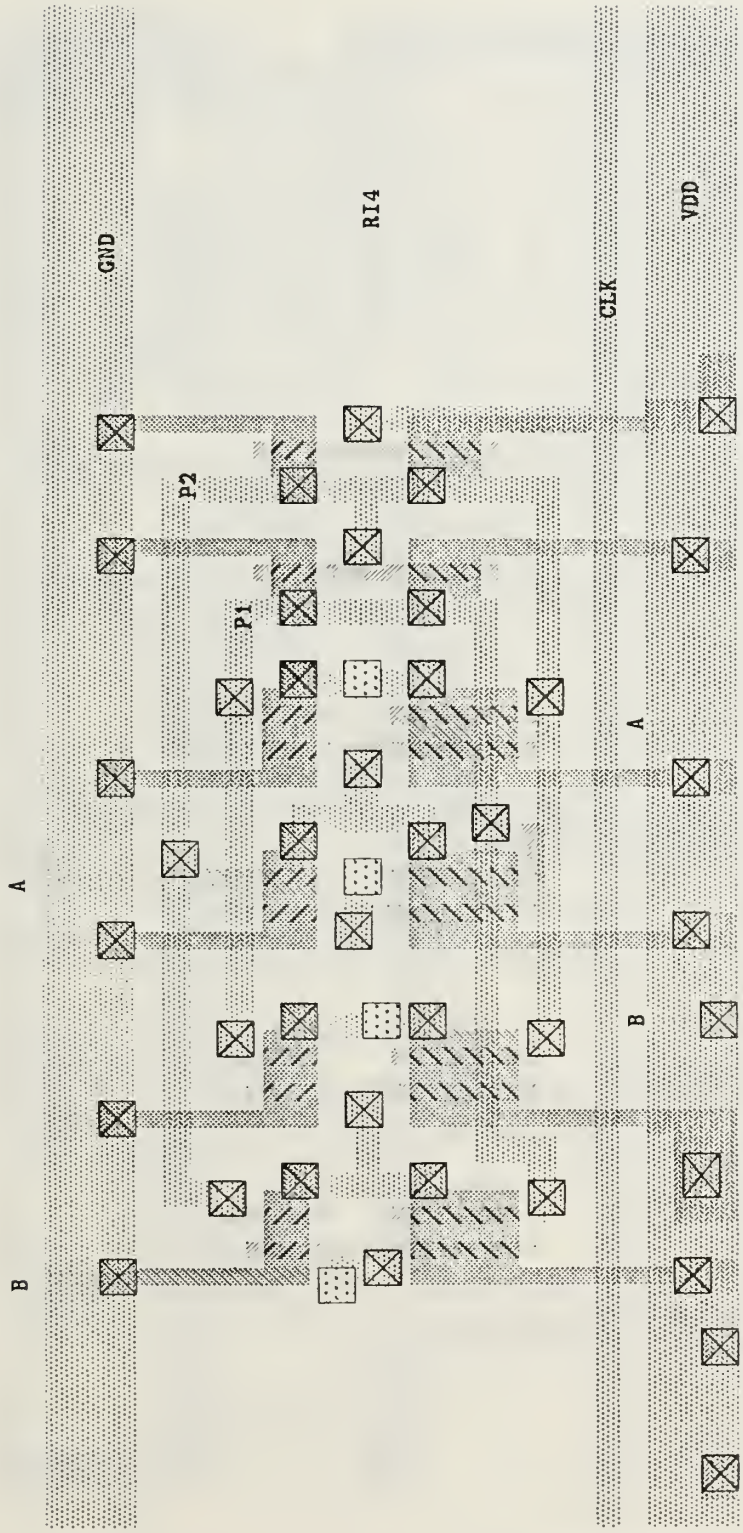


Figure 118. Cell RI4 layout

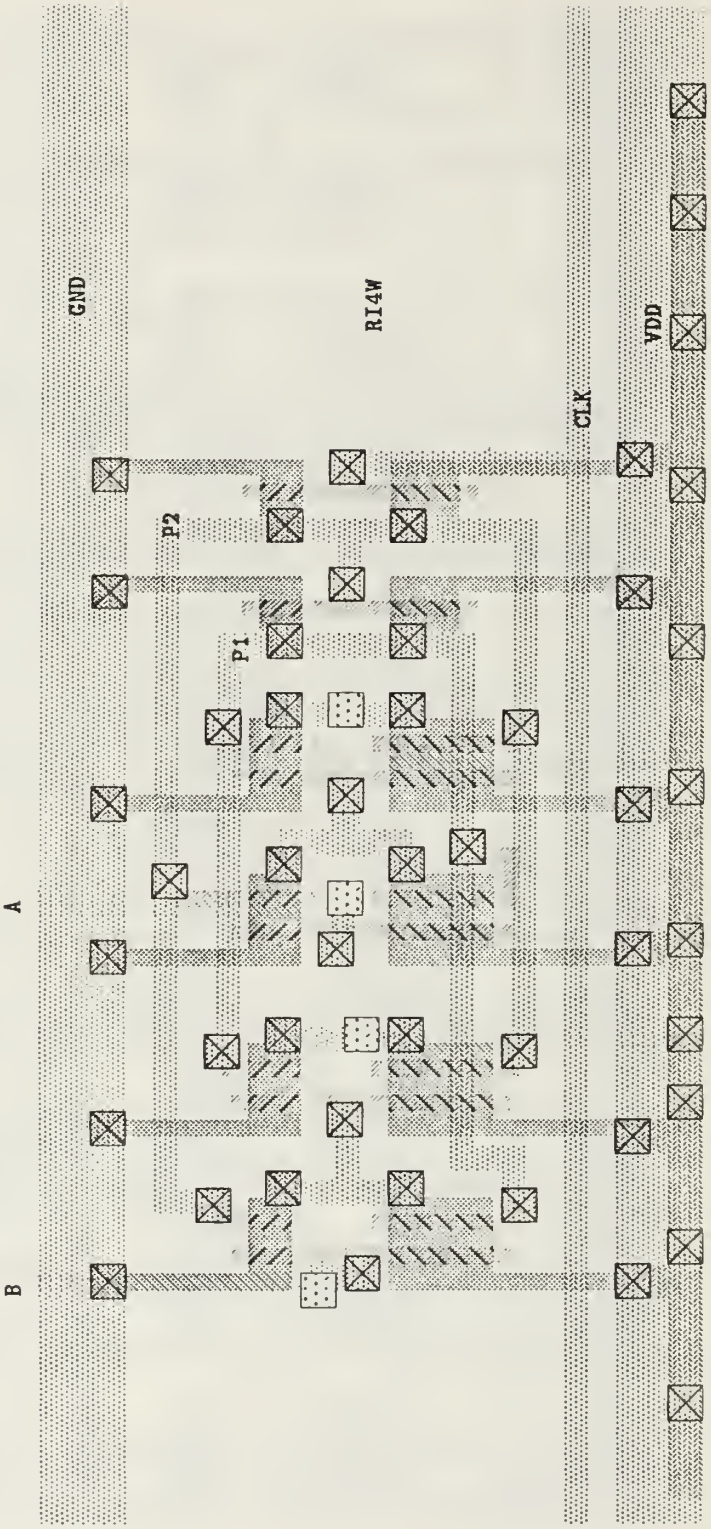


Figure 119. Cell RI4W layout

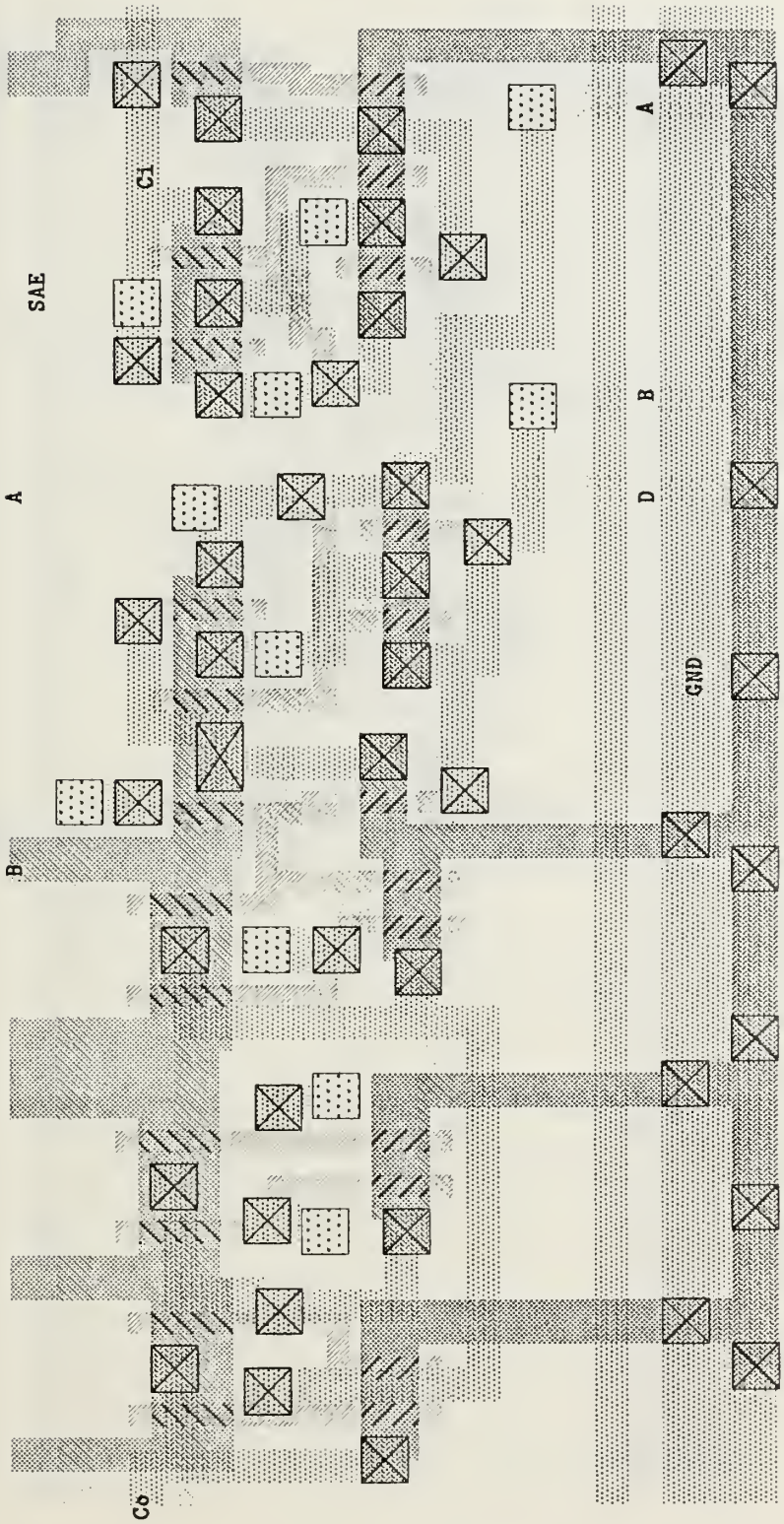


Figure 120. Cell SAE layout

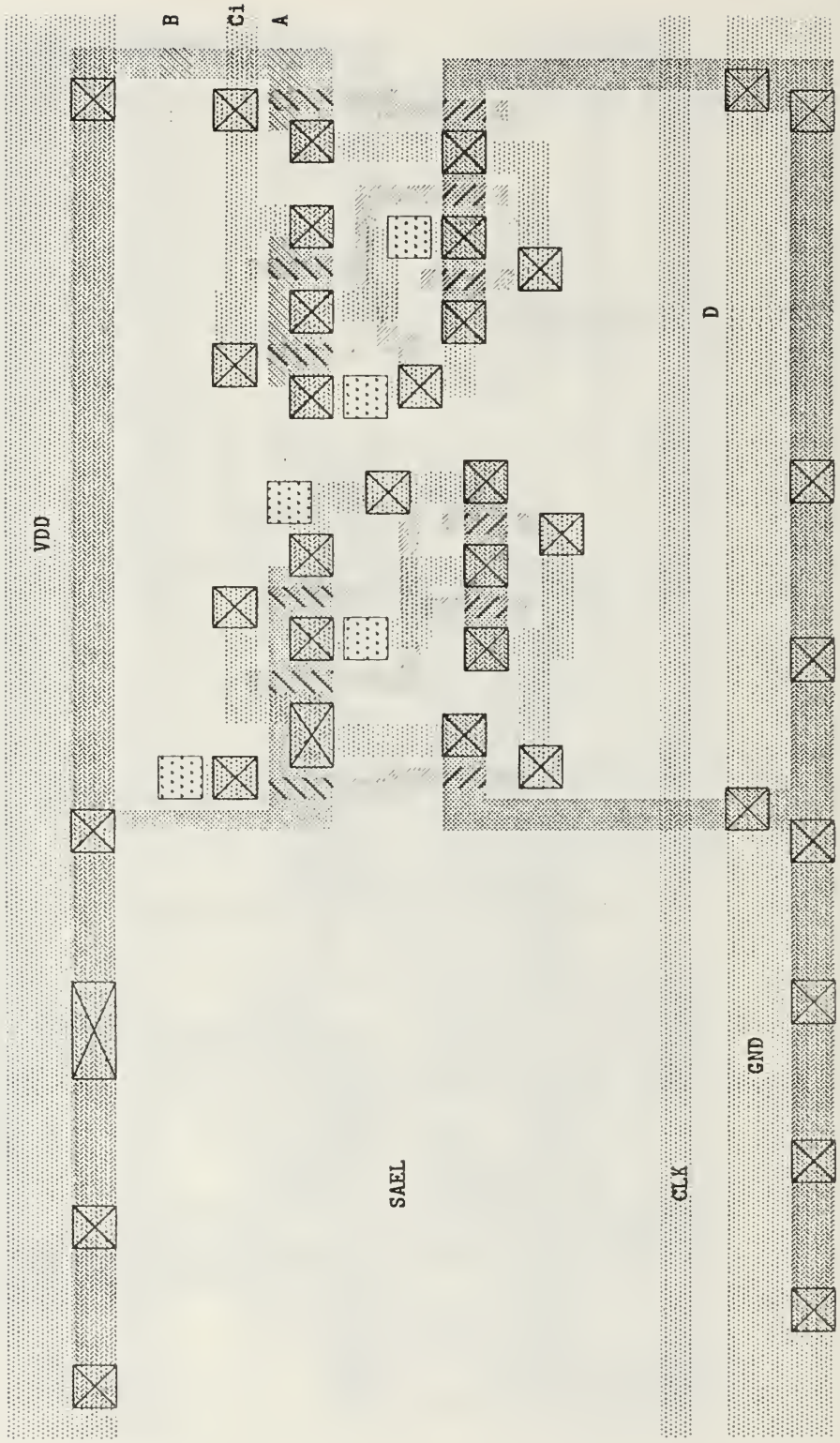


Figure 121. Cell SAEL layout

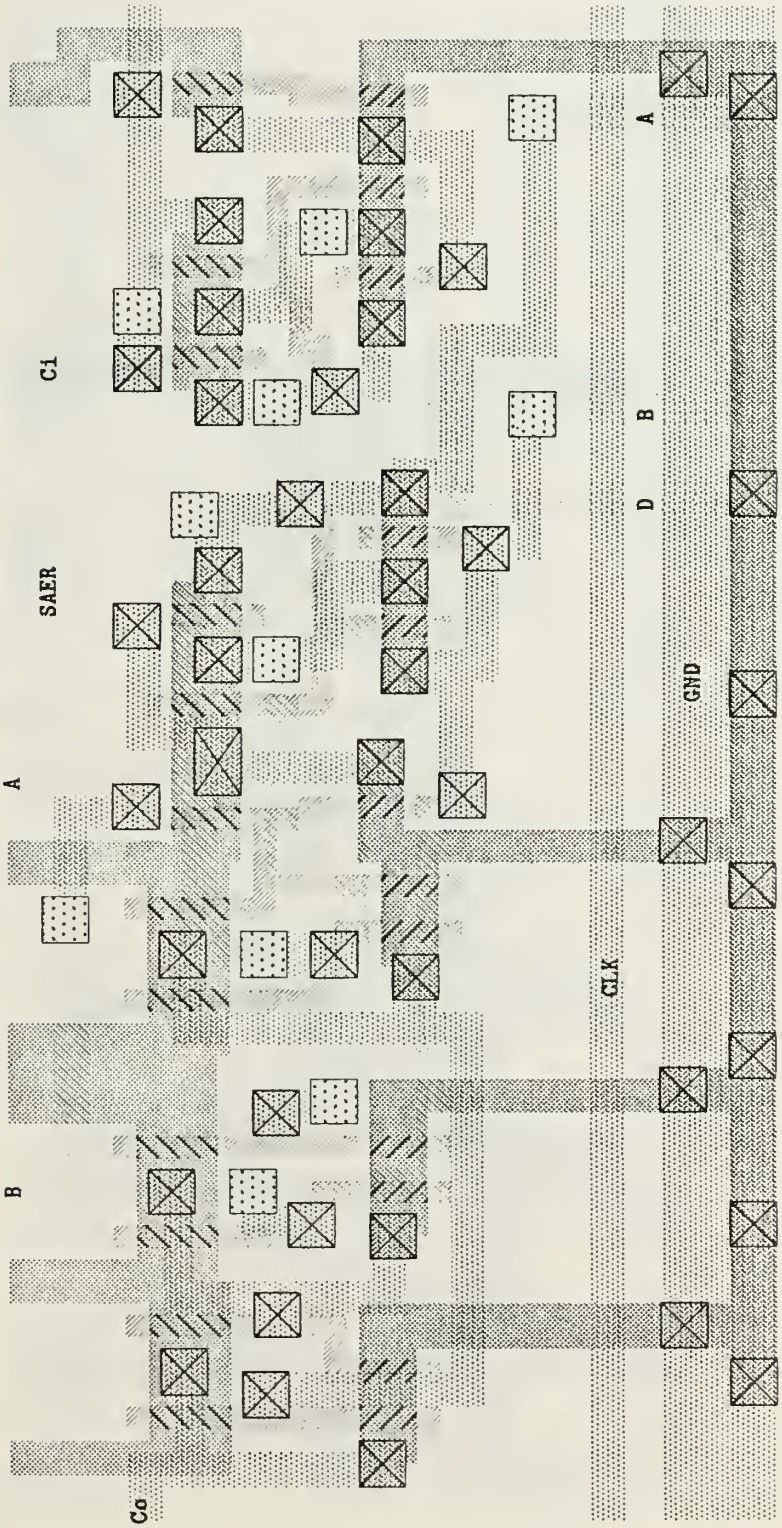


Figure 122. Cell SAER layout

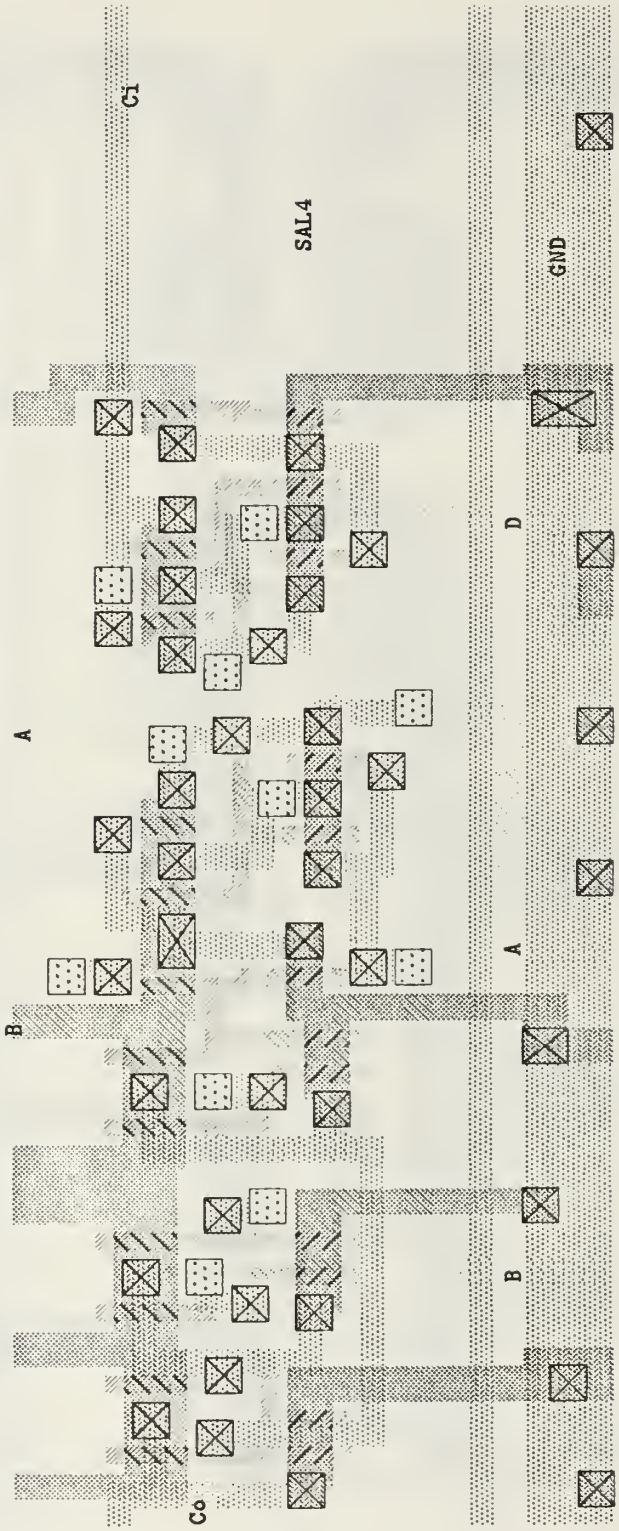


Figure 123. Cell SAL4 layout

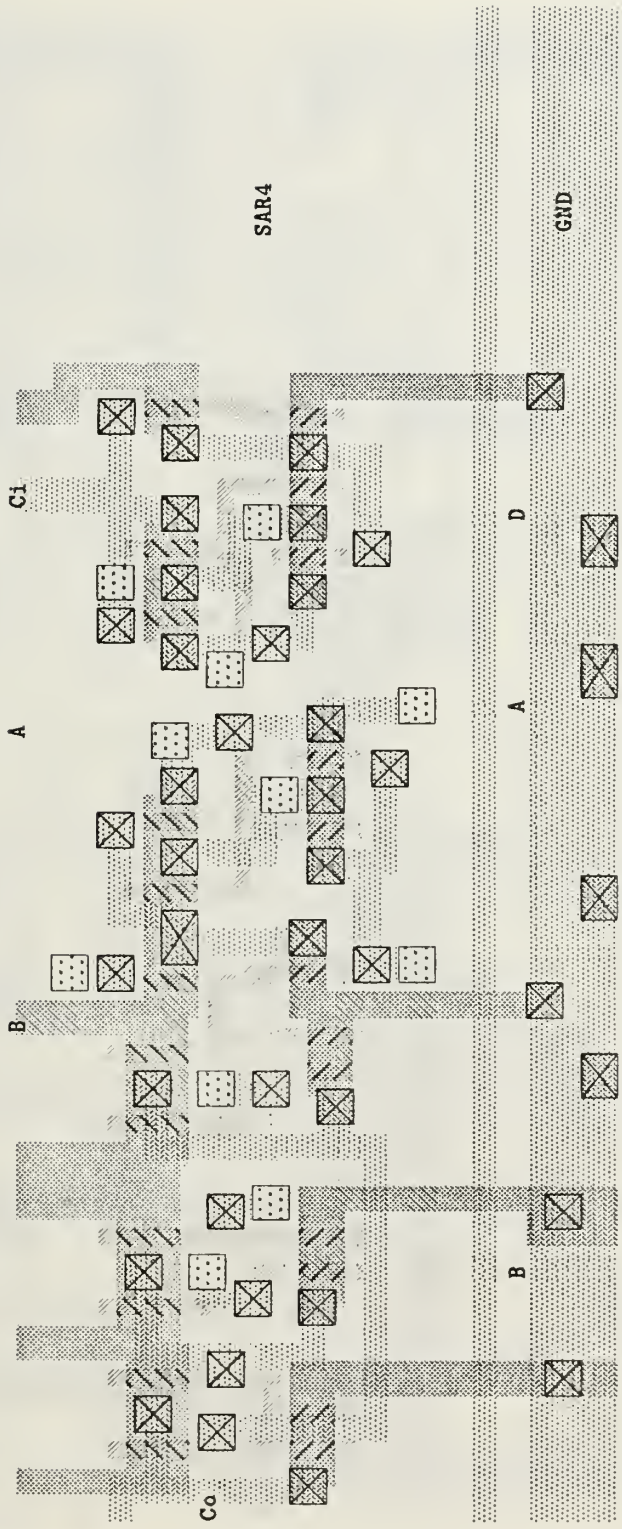


Figure 124. Cell SAR4 layout

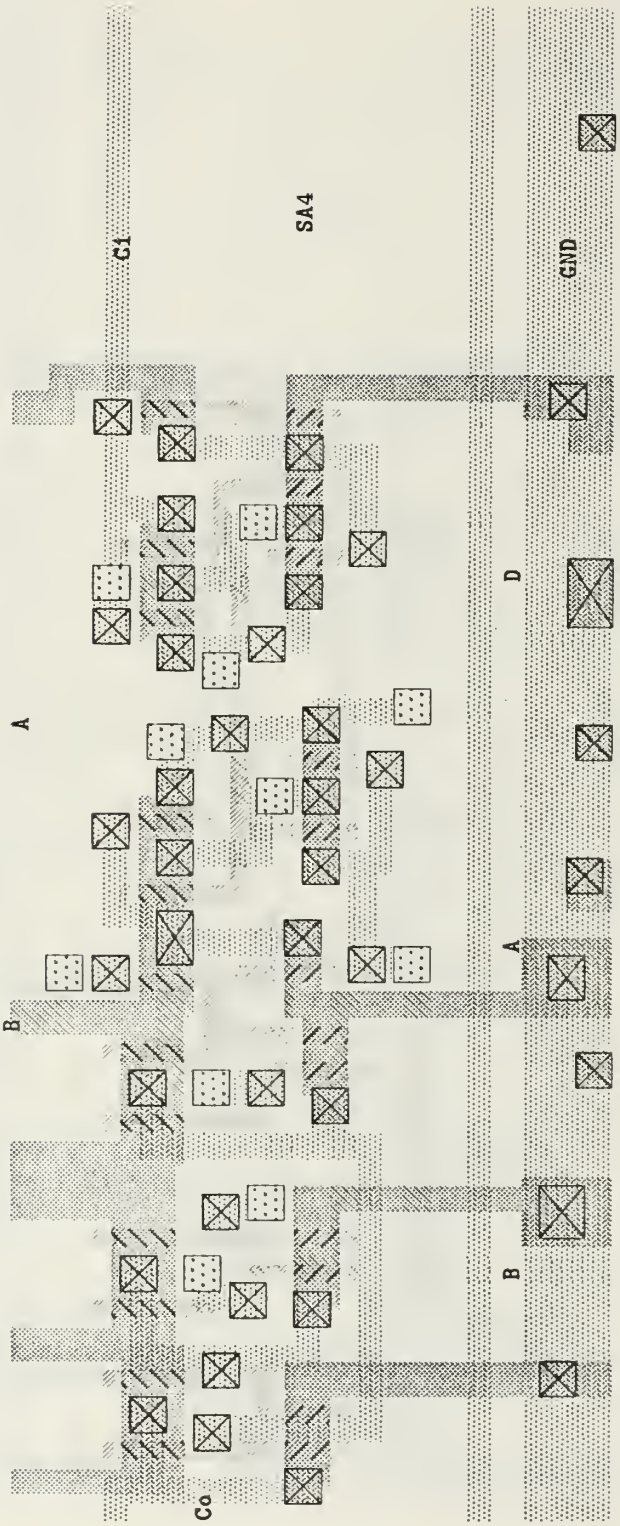


Figure 125. Cell SA4 layout

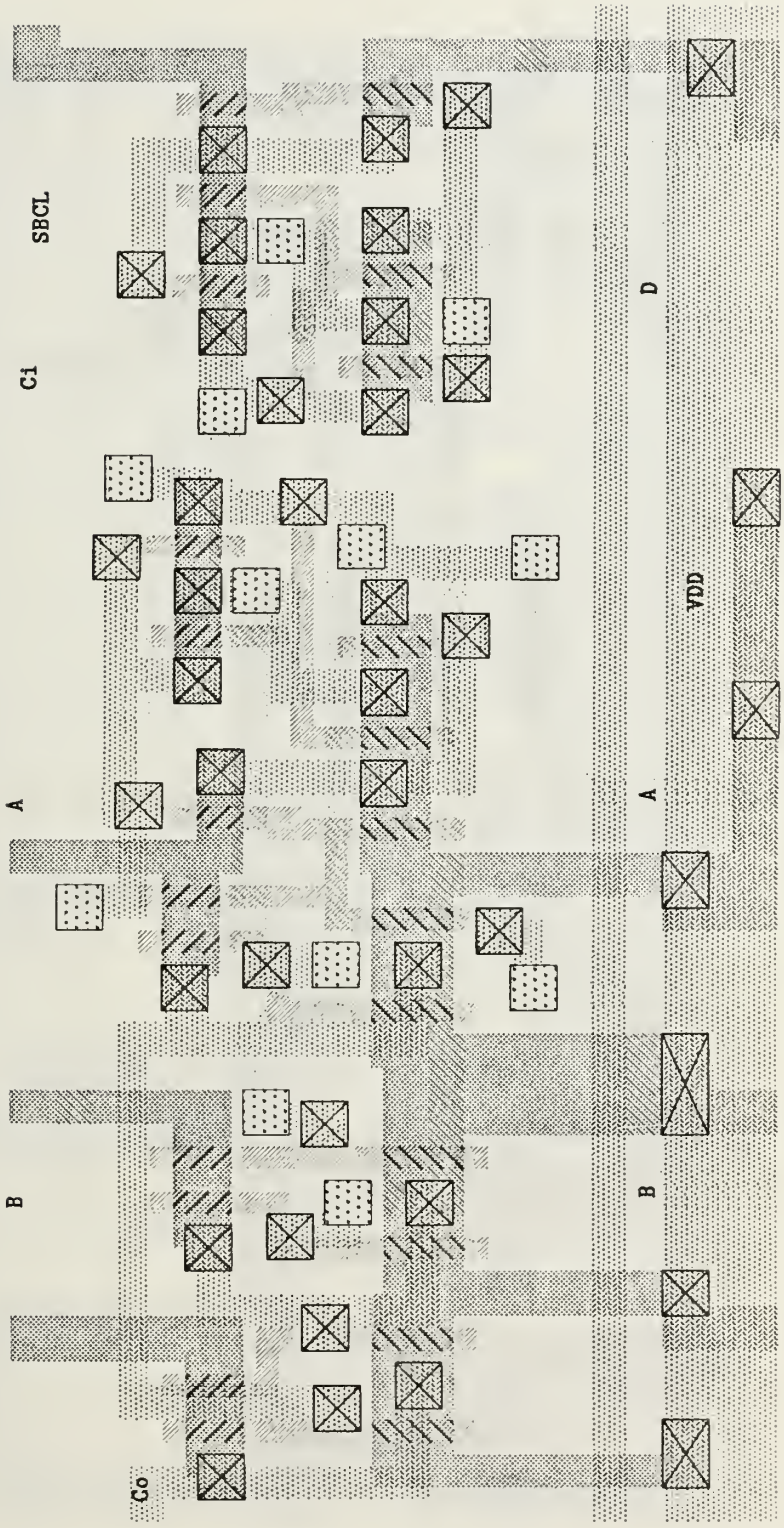


Figure 126. Cell SBCL layout

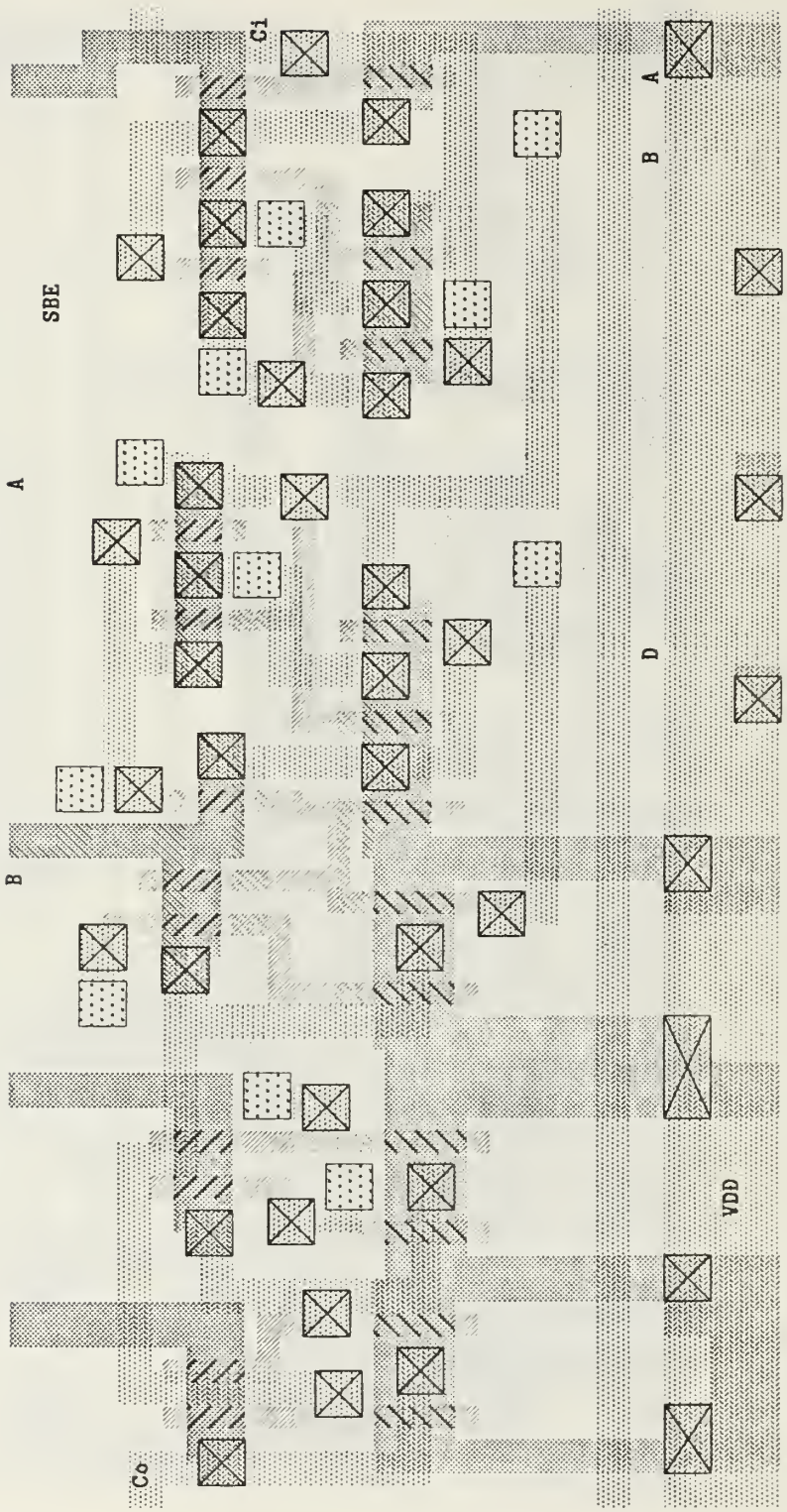


Figure 127. Cell SBE layout

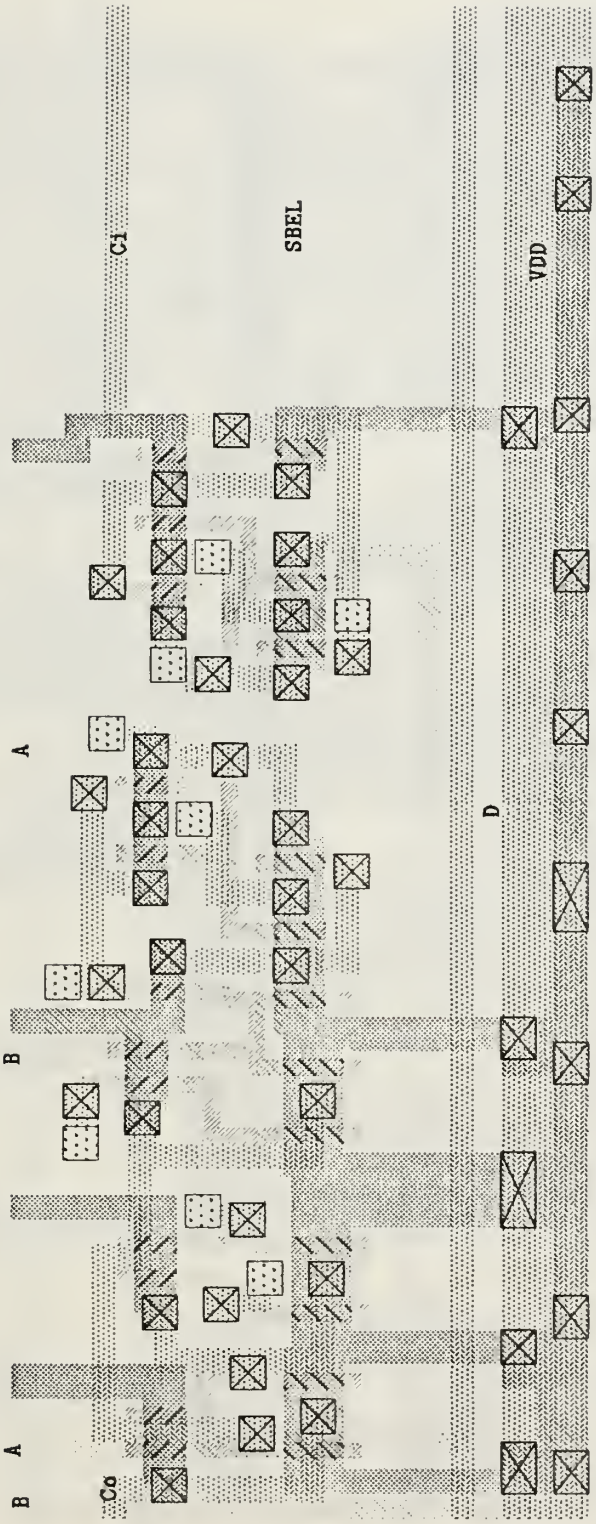


Figure 128. Cell SBEL layout

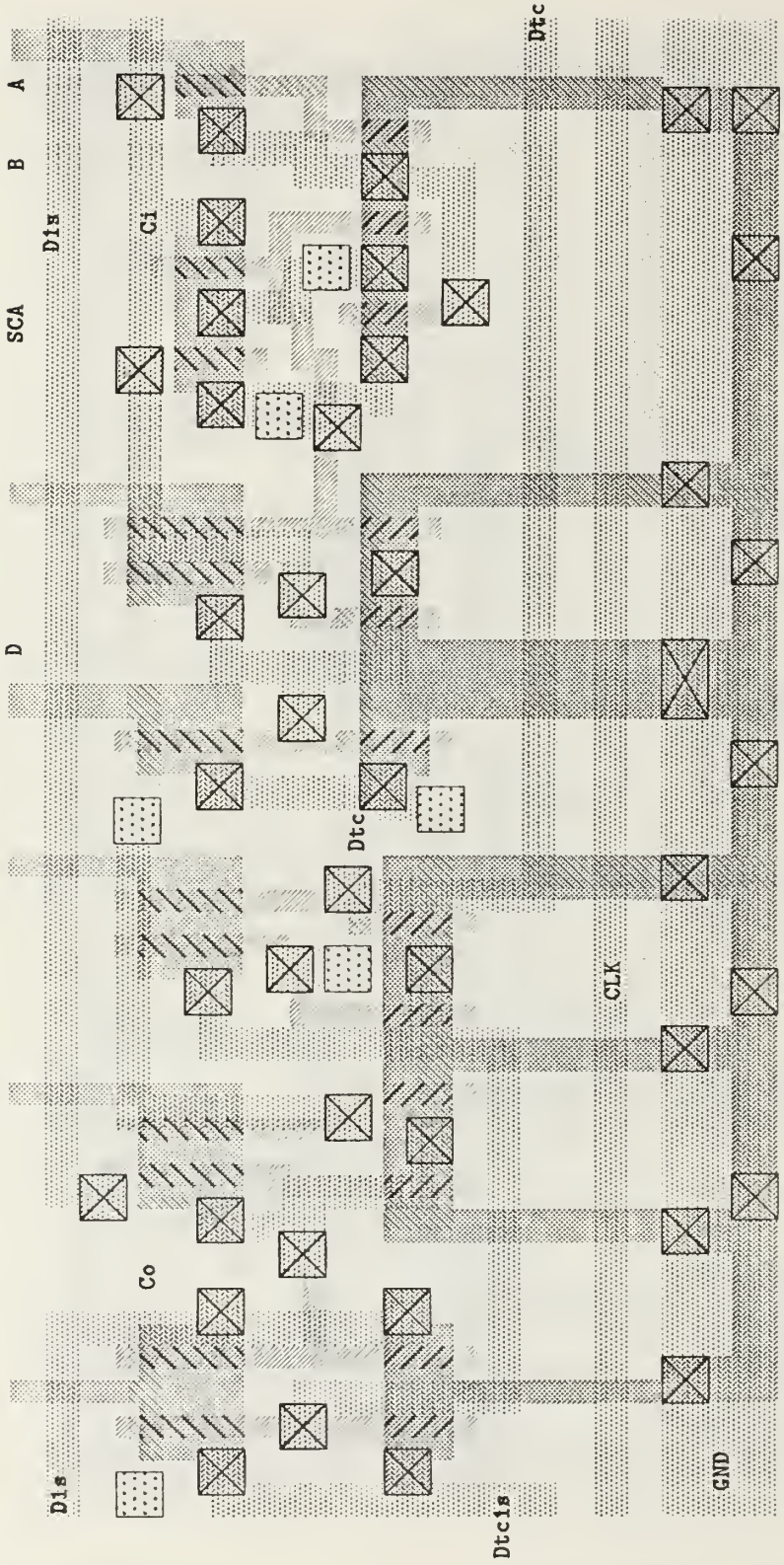


Figure 129. Cell SCA layout

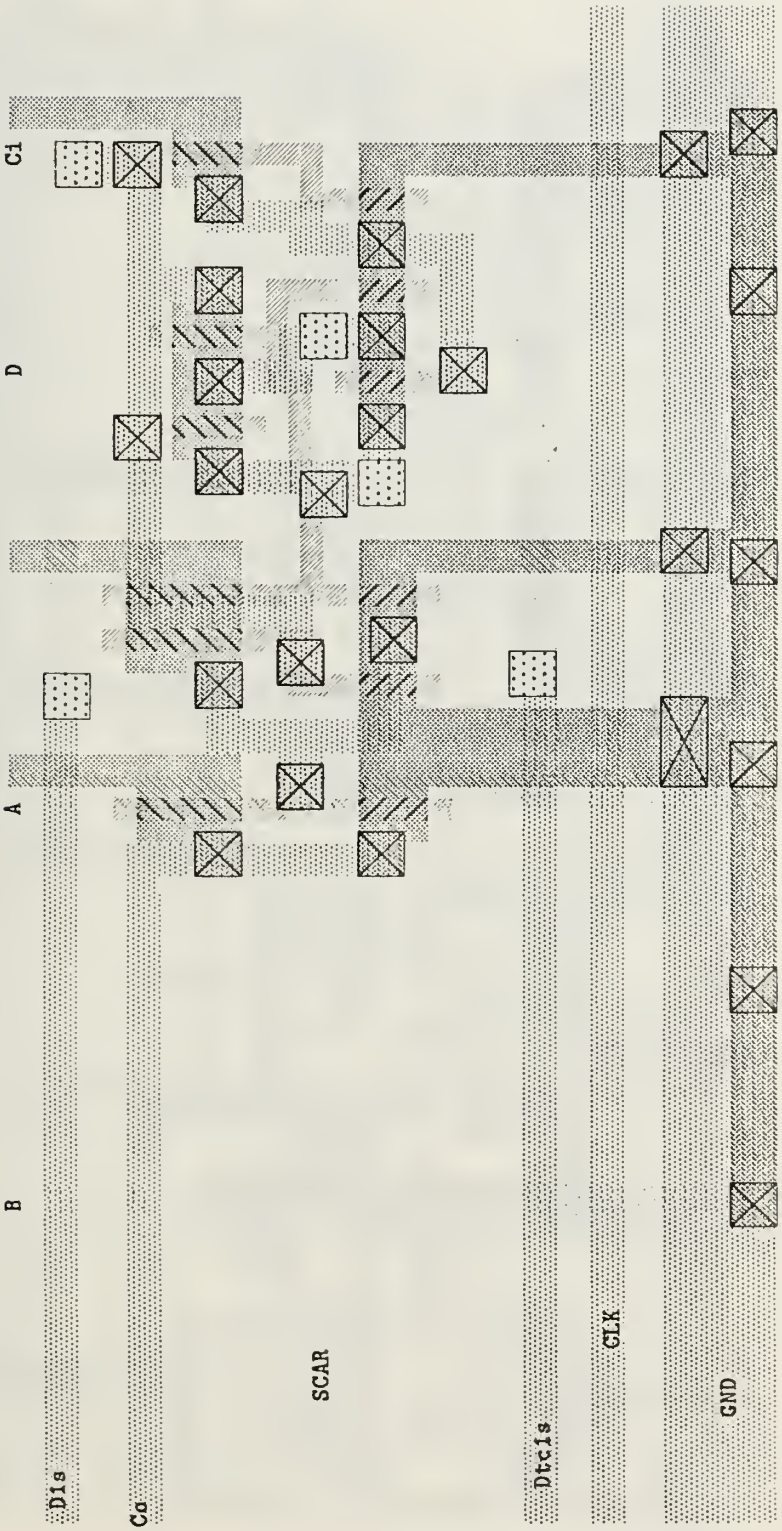


Figure 130. Cell SCAR layout

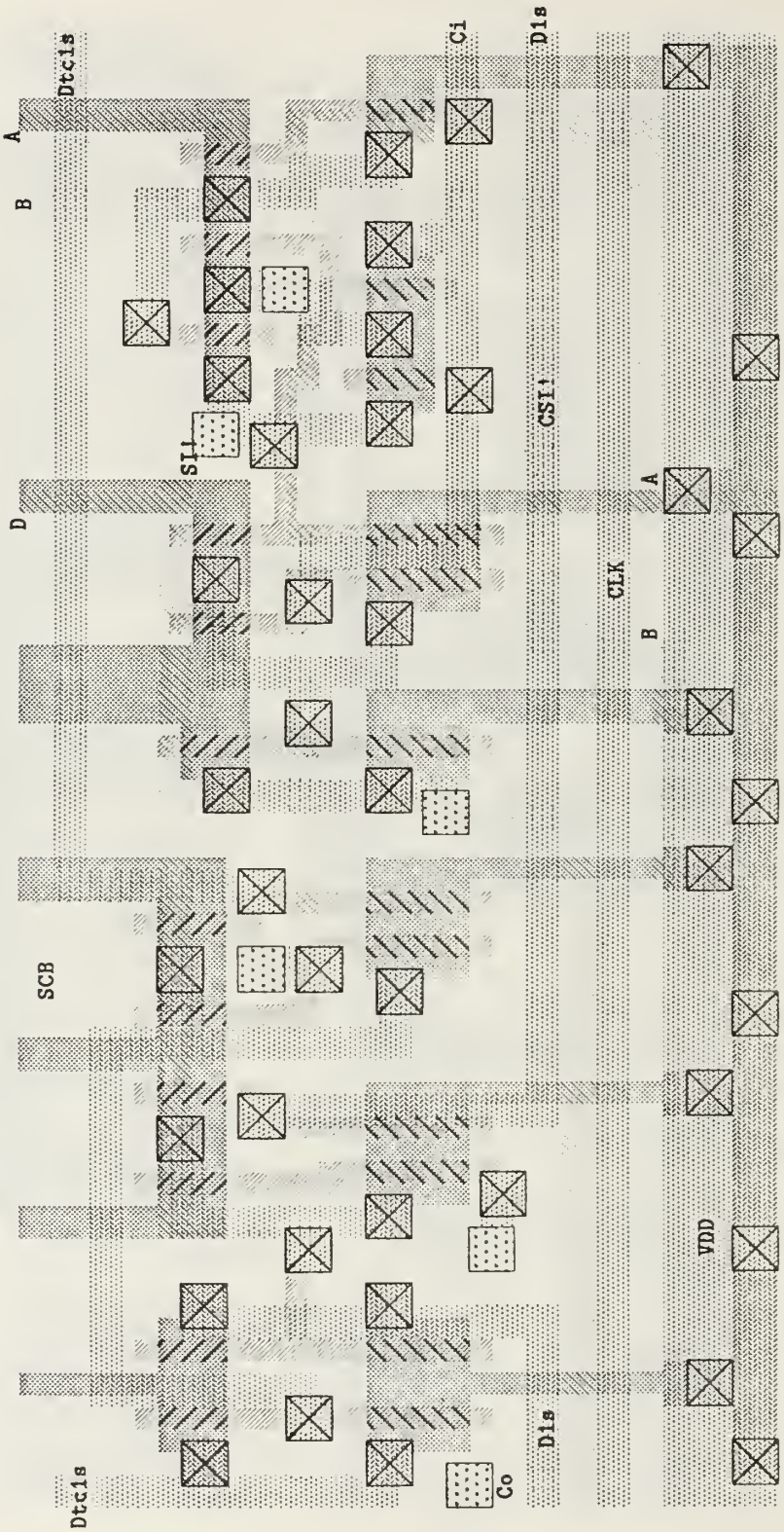


Figure 131. Cell SCB layout

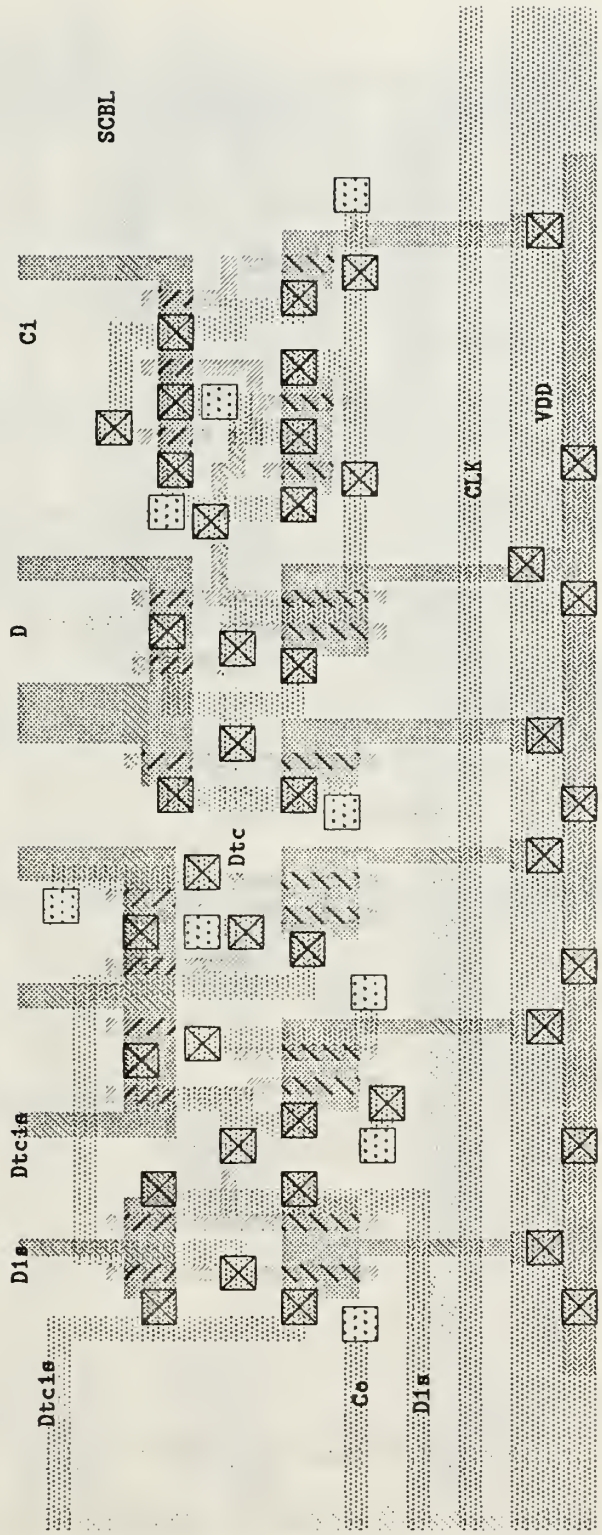


Figure 132. Cell SCBL layout

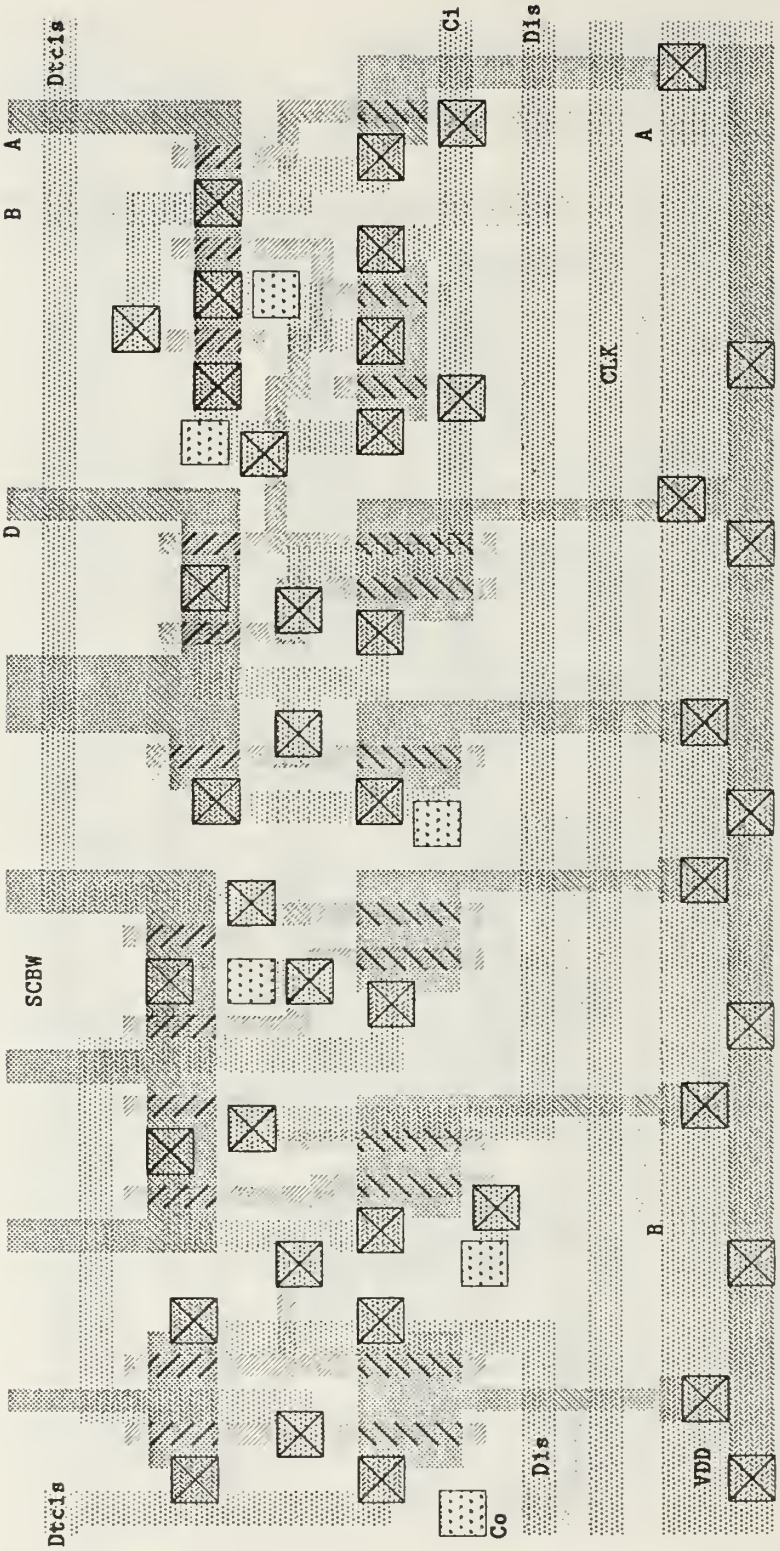


Figure 133. Cell SCBW layout

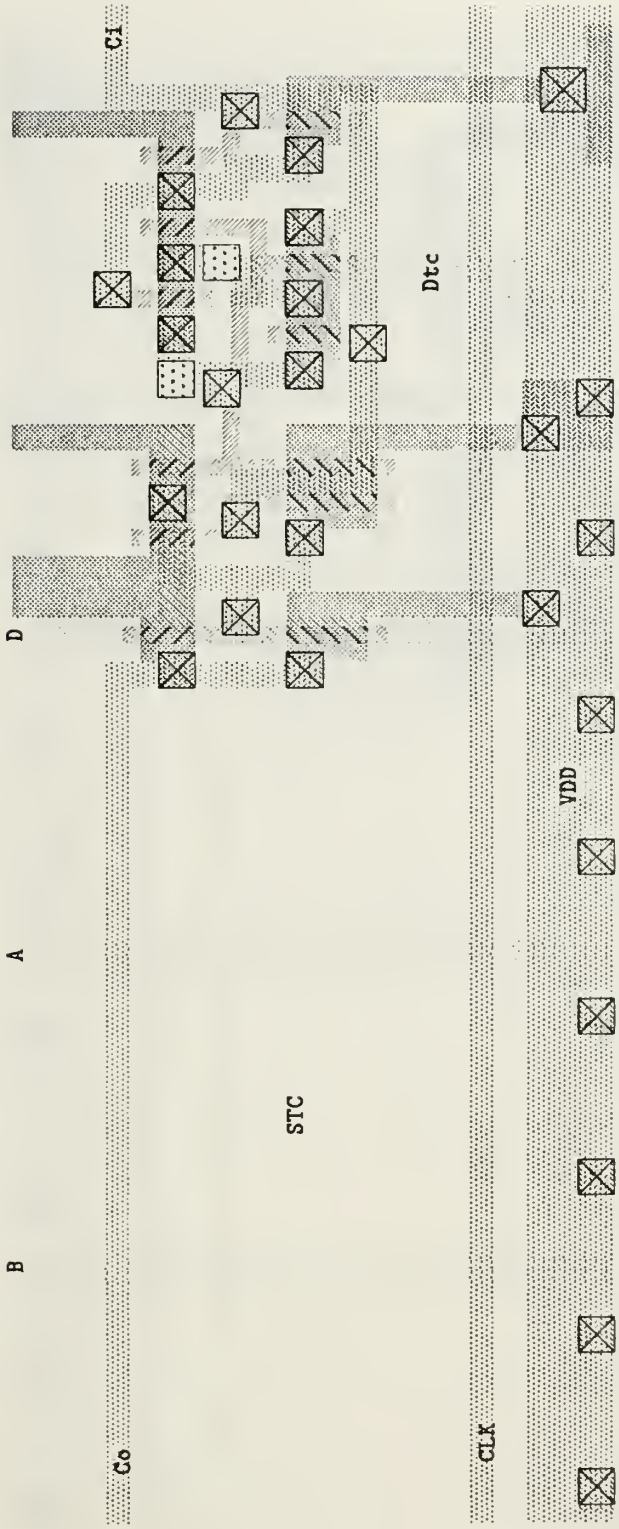


Figure 134. Cell STC layout

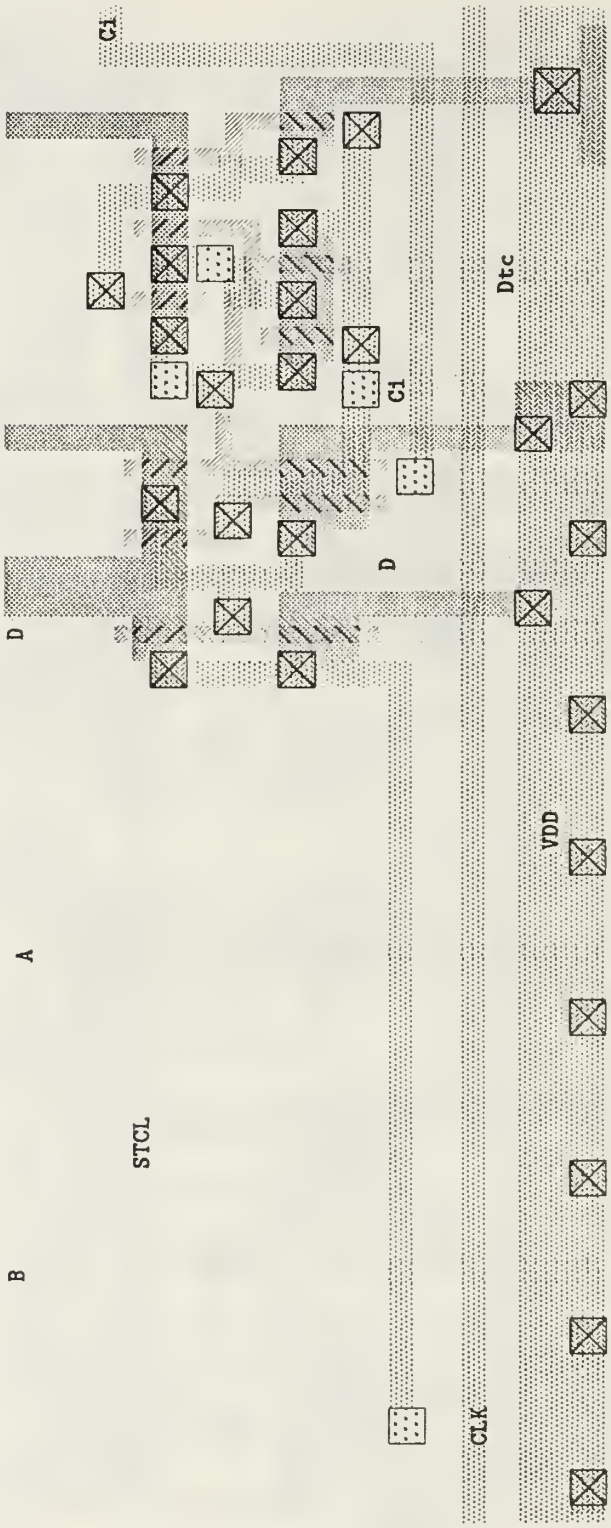


Figure 135. Cell STCL layout

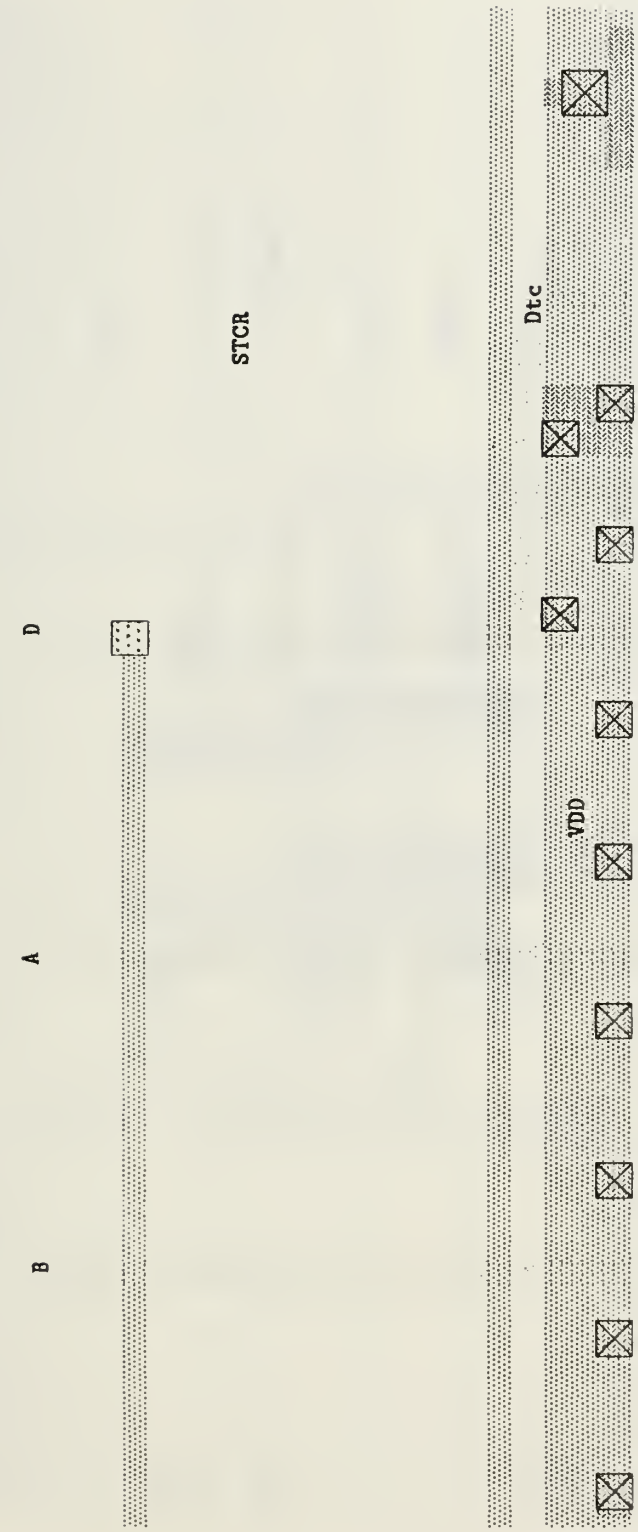


Figure 136. Cell STCR layout

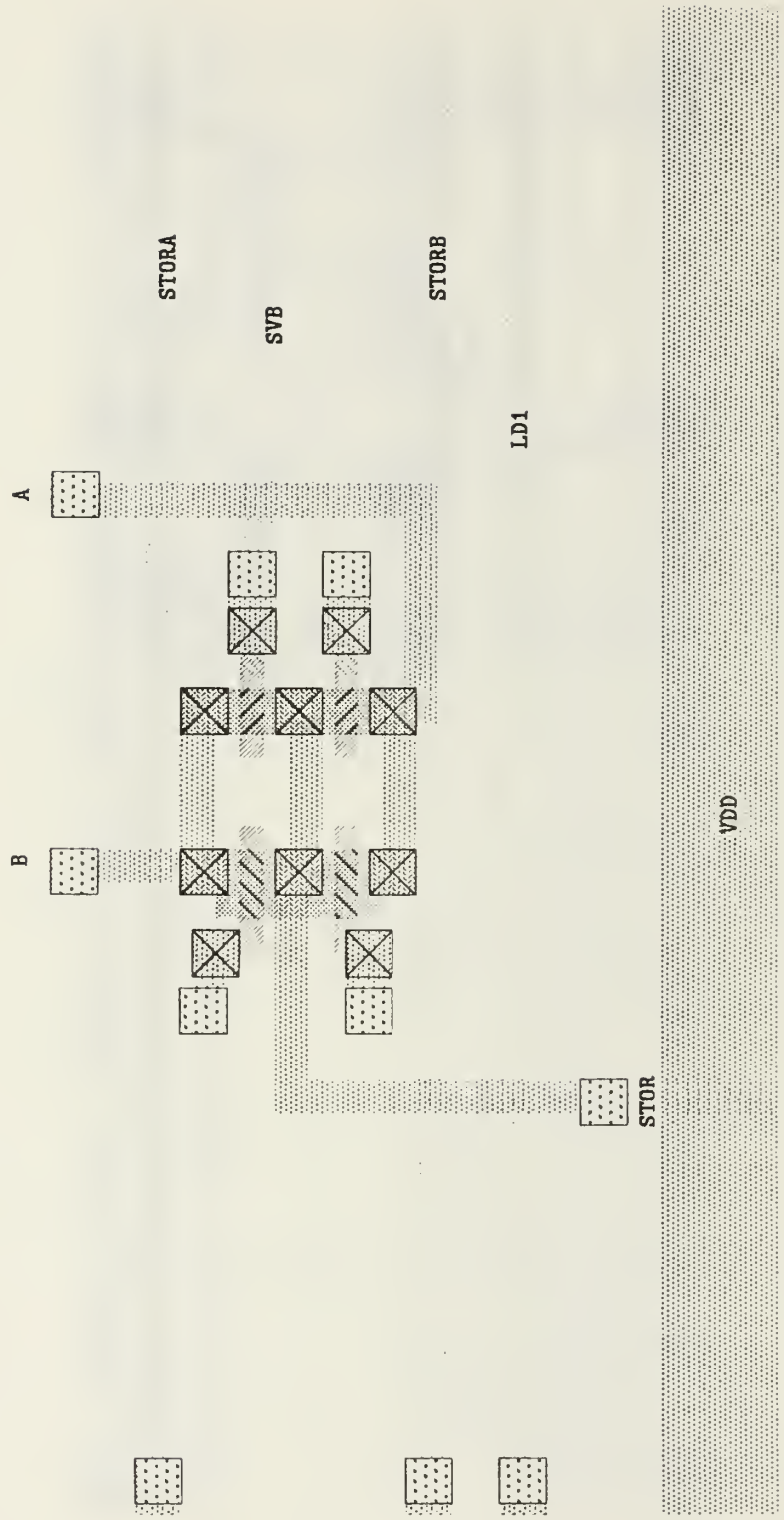


Figure 137. Cell SVB layout

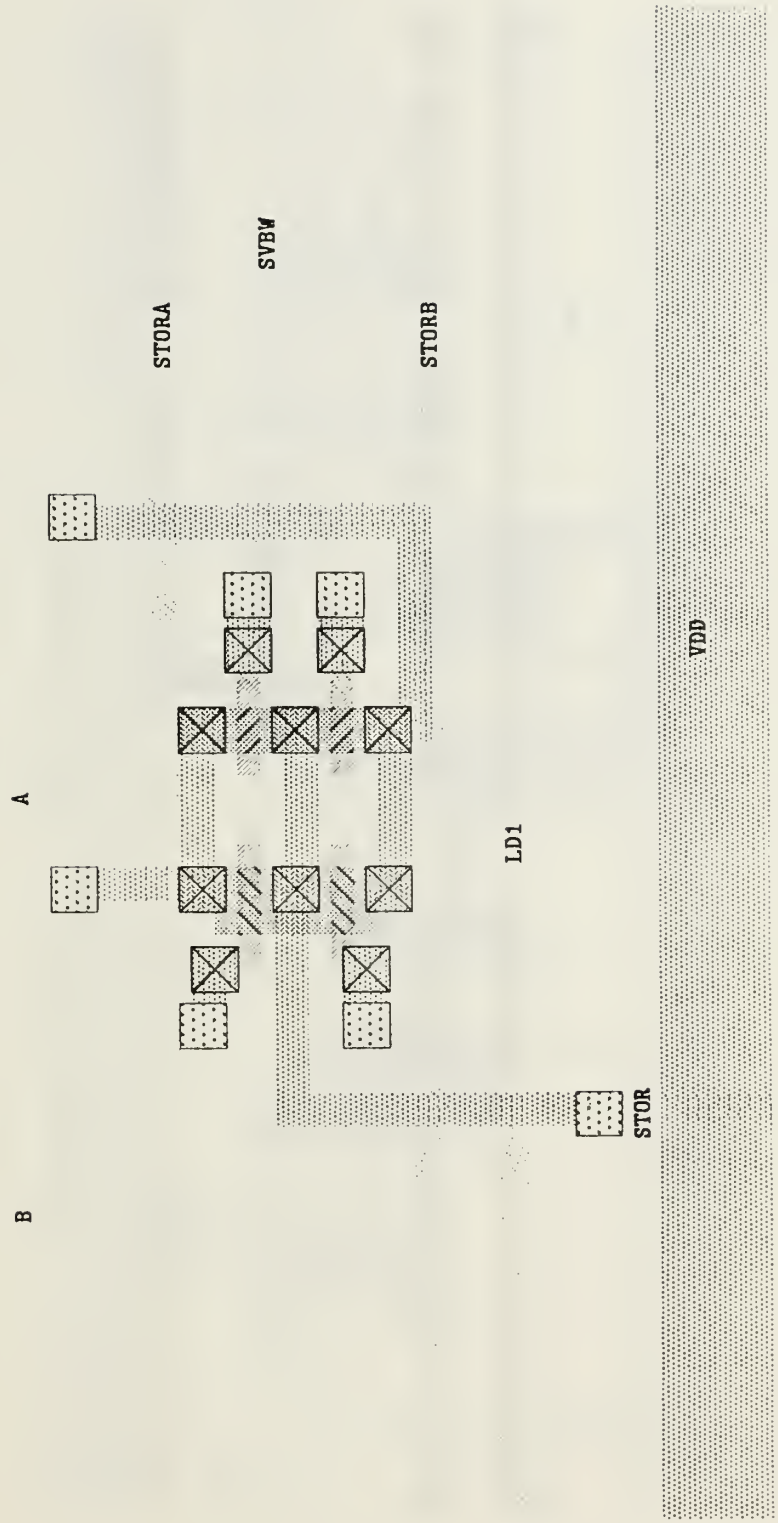


Figure 138. Cell SVBW layout

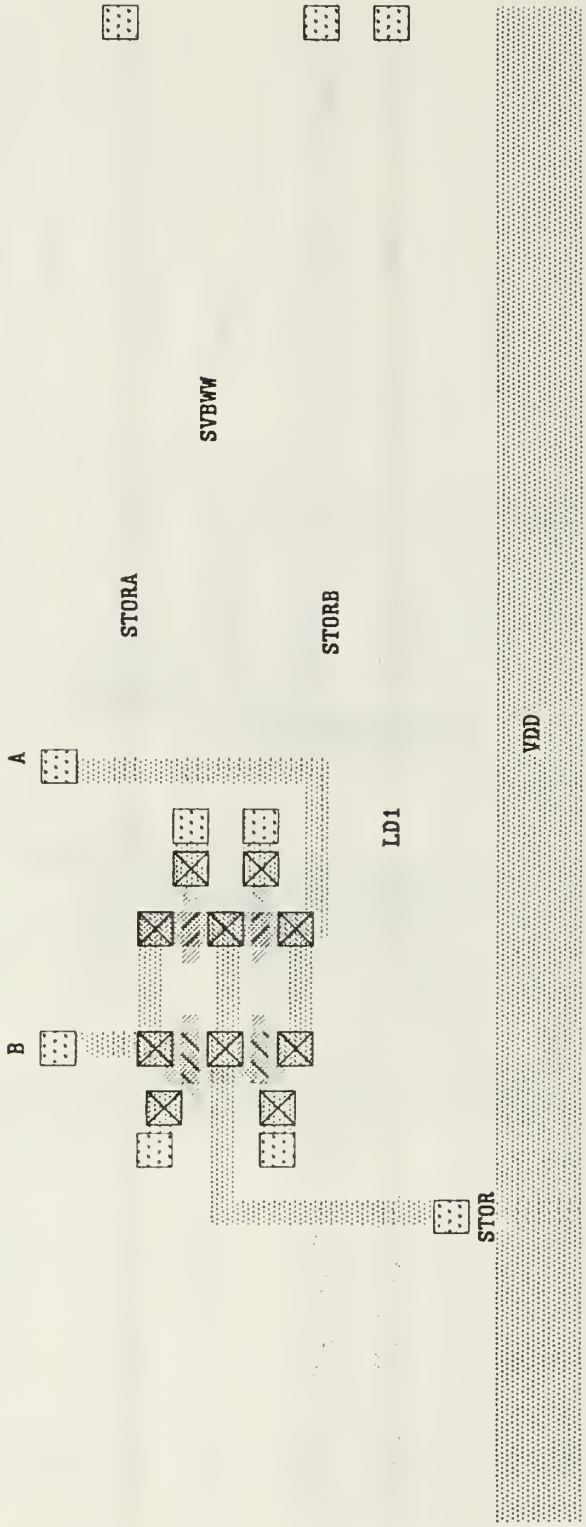


Figure 139. Cell SVBWW layout

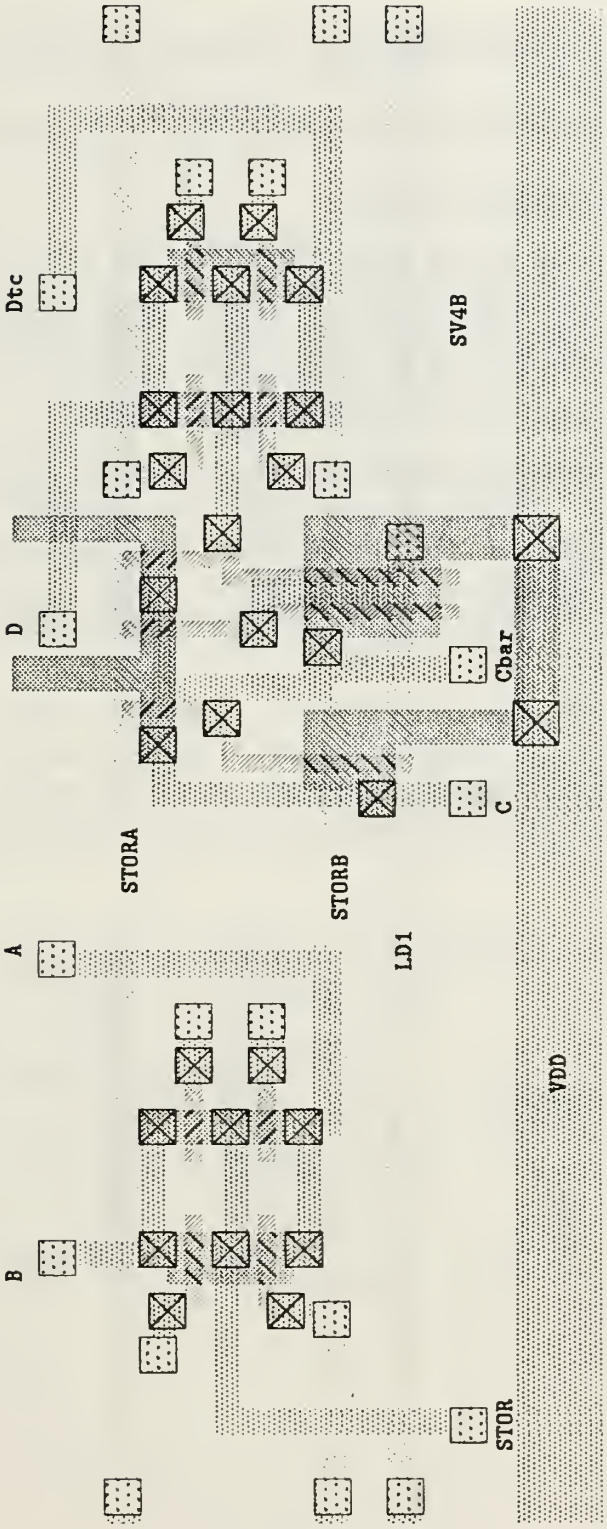


Figure 140. Cell SV4B layout

D. MANTISSA ALIGNMENT AND SELECTION

Most of the functions required to perform a floating point addition can be accomplished using cells previously documented. The mantissa selection and shift network cells are described below:

Cell SFAB: Figure 141 on page 189; a selection cell composed of eight transmission gates. The cell routes two pairs of mantissa bits to the storage and alignment cells as appropriate. The control line signals, STORA and STORB, are the signals generated by cell EXSLOG described in the exponent subtraction function cell list.

Cell SFI: A second stage shift cell that is composed of eight transmission gates and an inverter. Figure 33 on page 57 shows a gate level diagram of a cell and the control signal generation gates. Figure 142 on page 190 shows a layout of three SFI cells. For a 16-bit mantissa, 21 SFI cells must be placed next to each other to form the second stage shift network.

Cell SF5: A first stage shift cell composed of two transmission gates. Figure 32 on page 56 shows a gate level diagram of two SF5 cells and the control signal generation gates. Figure 143 on page 191 shows a layout of three SF5 cells placed next to each other.

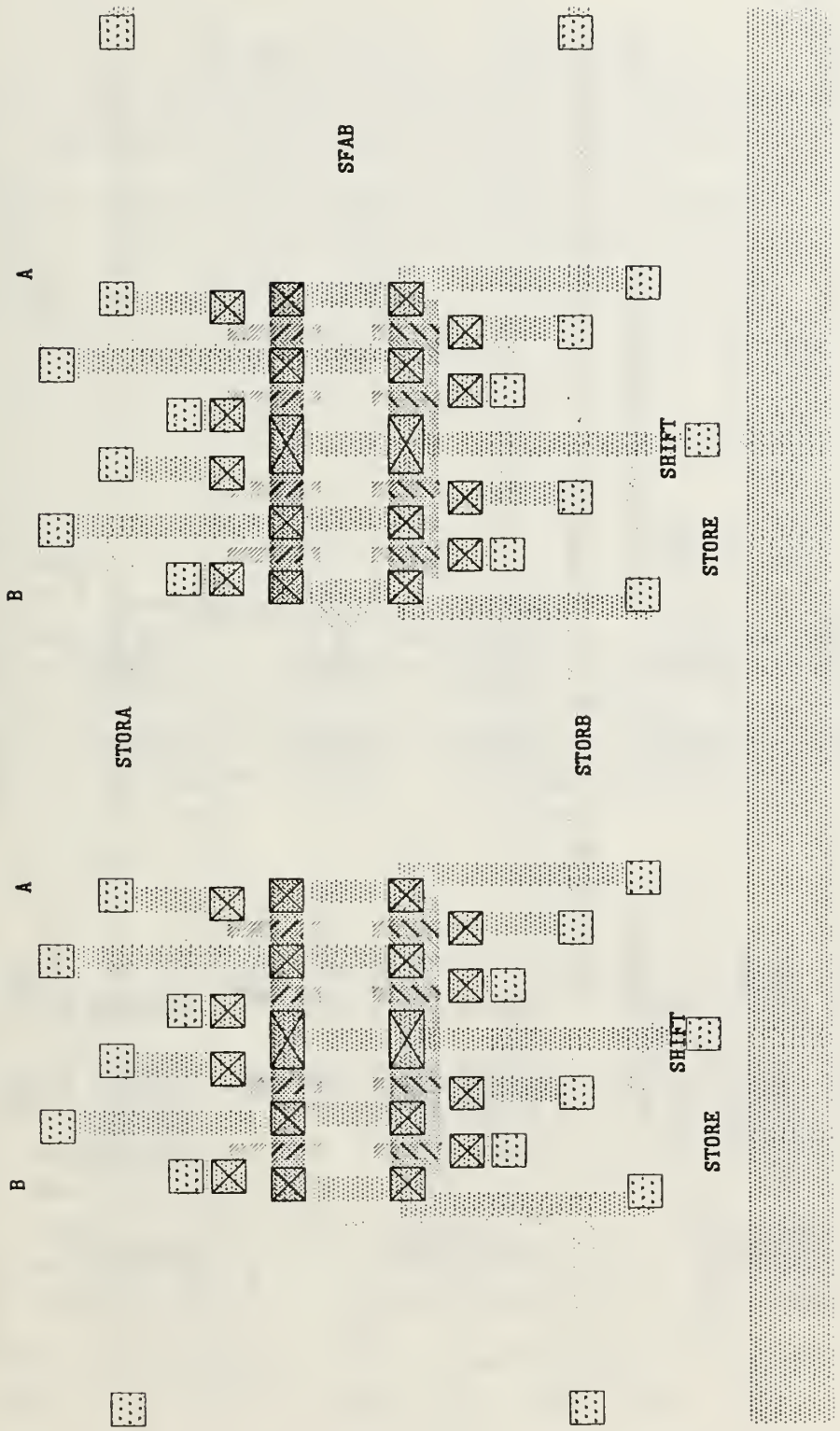


Figure 141. Cell SFAB layout

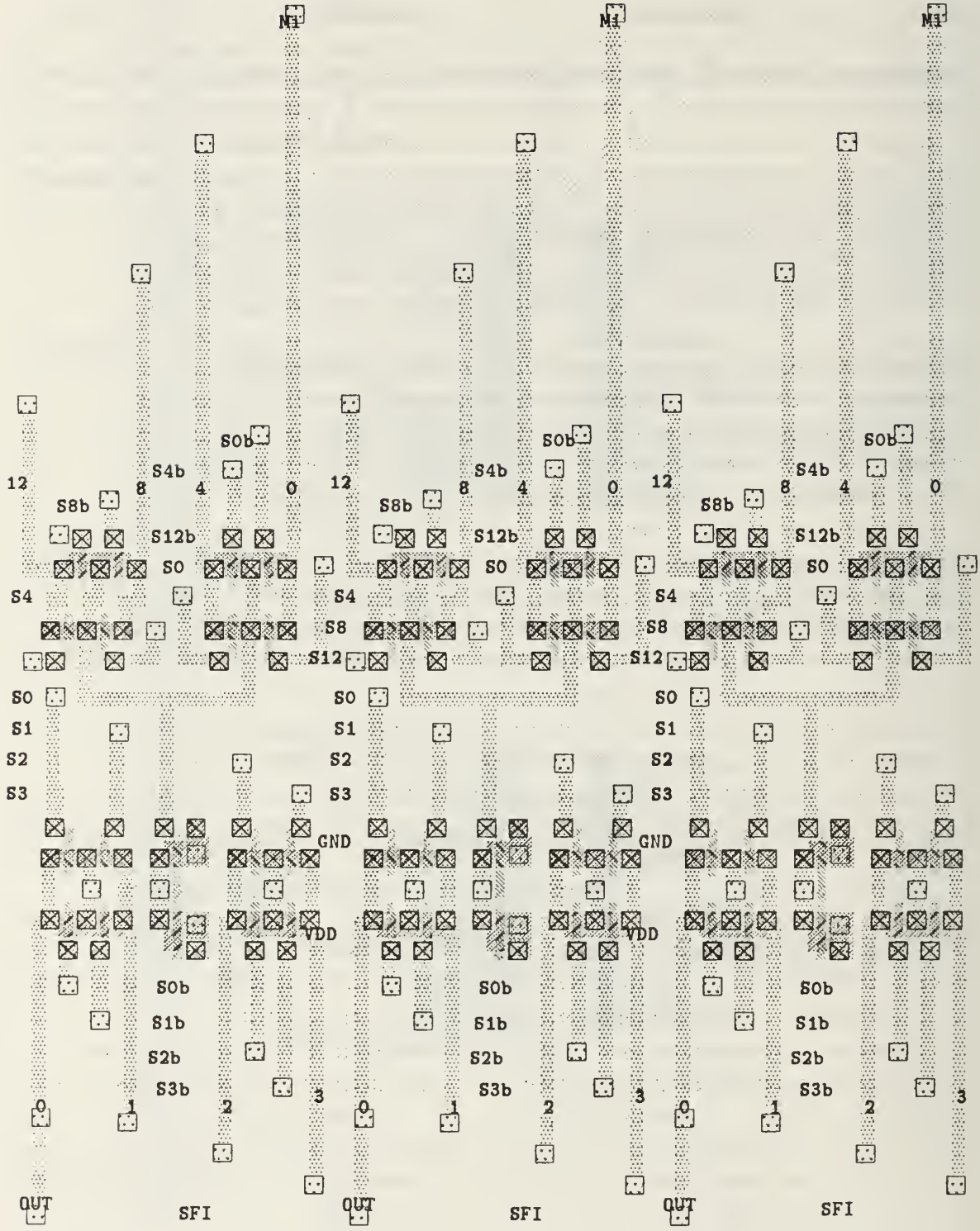


Figure 142. Cell SFI layout

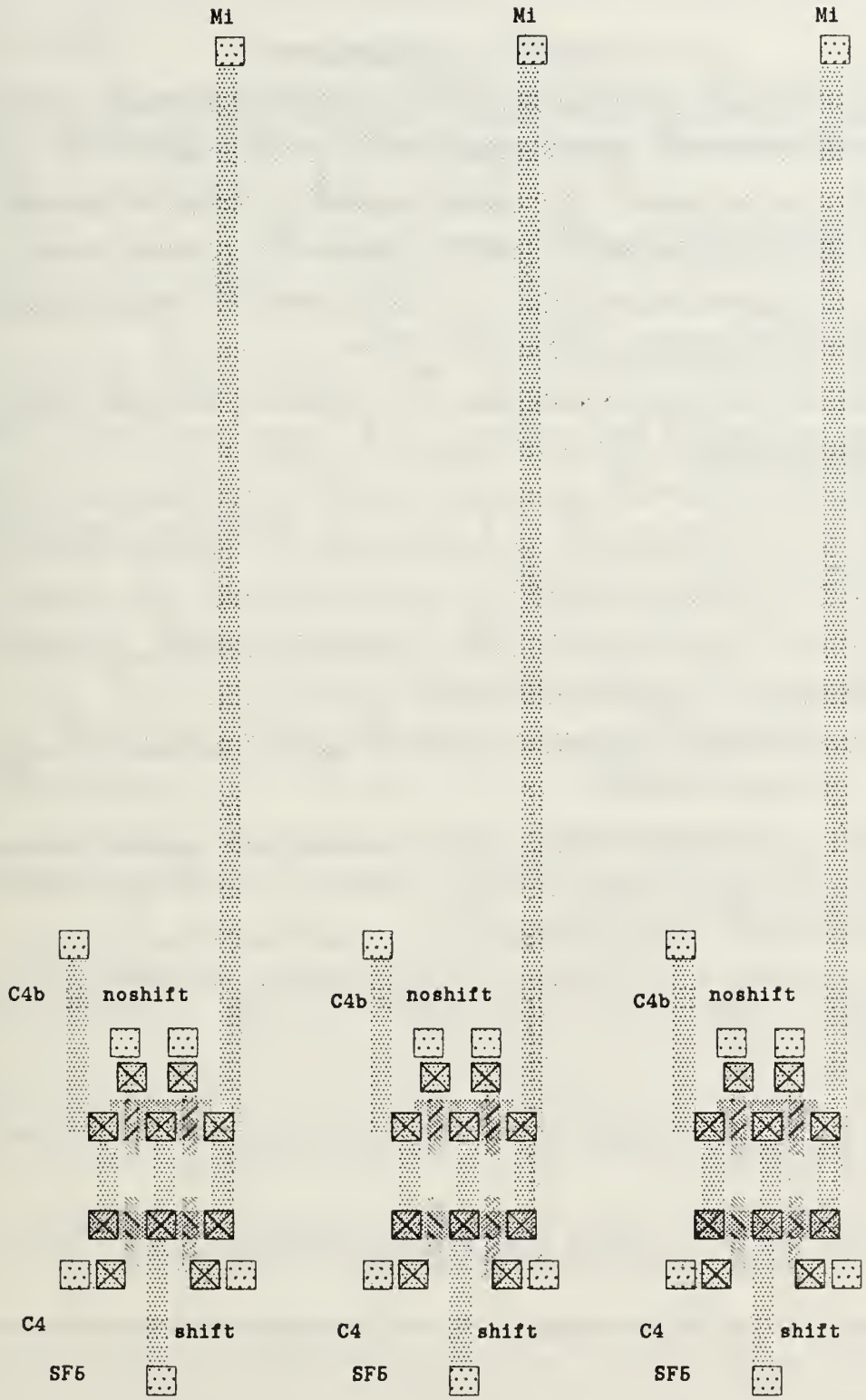


Figure 143. Cell SF5 layout

LIST OF REFERENCES

1. Randy S. Roberts, "Architectures for Digital Cyclic Spectral Analysis," Doctoral dissertation, University of California, Davis, September 1989

2. William A. Gardner, "The spectral correlation theory of cyclostationary time-series," *IEEE Signal Processing,* Vol. II, No. 1, pp. 13-36, 1986

3. R. S. Roberts and H. H. Loomis, Jr, "Digital architectures for estimating the cyclic cross spectrum," 1990 (unpublished)

4. Charles L. Rowe, Jr.,"Detection and analysis of direct sequence spread spectrum signals." Masters thesis, Naval Postgraduate School, Monterey, California, 1987

5. Robert D. Strum and Donald E. Kirk, *First Principles of Discrete Systems and Digital Signal Processing,* Addison-Wesley, Reading, Massachusetts, 1988

6. Harold S. Stone, *High-Performance Computer Architecture,* pp. 311-316, Addison-Wesley, Reading, Massachusetts, 1989

7. D. Payne, "Silicon Compilation in ASIC Design," *Defense Computing,* Vol. 1, No. 6, pp. 38-40, 1988

8. John Carl Davidson, "Implementation of a design for testability strategy using the Genesil Silicon Compiler," Master's thesis, Naval Postgraduate School, Monterey, California, 1989

9. Robert Howard Settle, "Design methodology using the Genesil Silicon Compiler," Masters thesis, Naval Postgraduate School, Monterey, California, 1988

10. R.R. Rockey, "Silicon compiler implementation of a Kalman filter algorithm as an ASIC," Masters thesis, Naval Postgraduate School, Monterey, California, 1988

11. Neil Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design,* Addison-Wesley, Reading, Massachusetts, 1988

12. Fang Lu and Henry Samueli, "A bit-level pipelined implementation of a CMOS multiplier-accumulator using a new pipelined full-adder cell design," *Proc. of 8th Annual International Phoenix Conf. on Computers and Communications,* pp. 45-65, IEEE Comput. Soc. Press, Washington, DC, Cat. No. 89CH2713-6, pp. 49-53, 1989

13. H. T. Kung, "Why systolic architectures?," *IEEE Computer,* pp. 37-46, January 1982

14. *1986 VLSI Tools: Still More Works by the Original Artists,* Walter S. Scott, Robert N. Mayo, Gordon Hamachi, and John K. Ousterhout, editors, Report No. UCB/CSD 86/272, Computer Science Division, University of California, Berkeley, California, December 1985

15. A. Vladimirescu, Kaihe Zhang, A. R. Newton, D. O. Pederson, and A Sangiovanni-Vincentelli, *SPICE User's Guide,* Northwest LIS Release 3.1, February, 1987

16. L. Howard Pollard, *Computer Design and Architecture,* Prentice Hall, New Jersey, 1990

17. Charles R. Raugh and Bruce A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. on Computers,* Vol. C-22, No.12, pp. 1045-1047, 1973

18. Joseph Y. Lee, Hugh L. Garvin, and Charles W. Slayman, "A high-speed high-density silicon 8X8-bit parallel multiplier," *IEEE Journal of Solid-State Circuits,* Vol. SC-22, No. 1, 1987

19. *Digital Signal Processing Applications with the TMS320 Family; Theory, Algorithms, and Implementations,* Texas Instruments SPRA012A, Houston, Texas, 1986

20. Herbert Herbert Taub, *Digital Circuits and Microprocessors,* McGraw-Hill, New York, 1982

21. A. Habibi, and P. A. Wintz, "Fast multipliers," *IEEE Transactions on Computers,* Vol. C-19, No. 2, pp. 153-157, August 1970

22. Kent P. Irwin, "Simulating transmission gate structures on Mossim II," Master's thesis, Naval Postgraduate School, Monterey, California, 1988

23. Ronald S. Huber, "Design of a pipelined multiplier using a silicon compiler," Master's thesis, Naval Postgraduate School, Monterey, California, 1990

24. *Genesil System Genport Users Guide,* Pub. No. 11-0084-2, Silicon Compiler Systems Corporation, San Jose, California, 1988

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curricular Officer, Code 32 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Prof. H. H. Loomis Jr., Code EC/Lm Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	5
6.	Prof. M. Cotton, Code EC/Cc Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
7.	Prof. Chyan Yang, Code EC/Ya Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
8.	Dr. W. A. Gardner Department of Electrical Engineering and Computer Science University of California Davis, California 95616	1